Pattern-based Segmentation of Digital Documents: Model and Implementation

Angelo Di Iorio

Technical Report UBLCS-2007-5

March 2007

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in PDF and gzipped PostScript formats via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`.

Recent Titles from the UBLCS Technical Report Series

2006-22 *Broadcasting at the Critical Threshold*, Arteconi, S., Hales, D., October 2006.

2006-23 *Emergent Social Rationality in a Peer-to-Peer System*, Marcozzi, A., Hales, D., October 2006.

2006-24 *Reconstruction of the Protein Structures from Contact Maps*, Margara, L., Vassura, M., di Lena, P., Medri, F., Fariselli, P., Casadio, R., October 2006.

2006-25 *Lambda Types on the Lambda Calculus with Abbreviations*, Guidi, F., November 2006.

2006-26 *FirmNet: The Scope of Firms and the Allocation of Task in a Knowledge-Based Economy*, Mollona, E., Marcozzi, A. November 2006.

2006-27 *Behavioral Coalition Structure Generation*, Rossi, G., November 2006.

2006-28 *On the Solution of Cooperative Games*, Rossi, G., December 2006.

2006-29 *Motifs in Evolving Cooperative Networks Look Like Protein Structure Networks*, Hales, D., Arteconi, S., December 2006.

2007-01 *Extending the Choquet Integral*, Rossi, G., January 2007.

2007-02 *Towards Cooperative, Self-Organised Replica Management*, Hales, D., Marcozzi, A., Cortese, G., February 2007.

2007-03 *A Model and an Algebra for Semi-Structured and Full-Text Queries (PhD Thesis)*, Buratti, G., March 2007.

2007-04 *Data and Behavioral Contracts for Web Services (PhD Thesis)*, Carpineti, S., March 2007.

2007-05 *Pattern-Based Segmentation of Digital Documents: Model and Implementation (PhD Thesis)*, Di Iorio, A., March 2007.

2007-06 *A Communication Infrastructure to Support Knowledge Level Agents on the Web (PhD Thesis)*, Guidi, D., March 2007.

2007-07 *Formalizing Languages for Service Oriented Computing (PhD Thesis)*, Guidi, C., March 2007.

2007-08 *Secure Gossiping Techniques and Components (PhD Thesis)*, Jesi, G., March 2007.

2007-09 *Rich Media Content Adaptation in E-Learning Systems (PhD Thesis)*, Mirri, S., March 2007.

2007-10 *User Interaction Widgets for Interactive Theorem Proving (PhD Thesis)*, Zacchiroli, S., March 2007.

2007-11 *An Ontology-based Approach to Define and Manage B2B Interoperability (PhD Thesis)*, Gessa, N., March 2007.

2007-12 *Decidable and Computational Properties of Cellular Automata (PhD Thesis)*, Di Lena, P., March 2007.

Pattern-based Segmentation of Digital Documents: Model and Implementation

Angelo Di Iorio

Technical Report UBLCS-2007-5

March 2007

Abstract

This thesis proposes a new document model, according to which any document can be segmented in some independent components and transformed in a pattern-based projection, that only uses a very small set of objects and composition rules. The point is that such a normalized document expresses the same fundamental information of the original one, in a simple, clear and unambiguous way.

The central part of my work consists of discussing that model, investigating how a digital document can be segmented, and how a segmented version can be used to implement advanced tools of conversion. I present seven patterns which are versatile enough to capture the most relevant documents' structures, and whose minimality and rigour make that implementation possible.

The abstract model is then instantiated into an actual markup language, called IML. IML is a general and extensible language, which basically adopts an XHTML syntax, able to capture a posteriori the only content of a digital document. It is compared with other languages and proposals, in order to clarify its role and objectives.

Finally, I present some systems built upon these ideas. These applications are evaluated in terms of users' advantages, workflow improvements and impact over the overall quality of the output. In particular, they cover heterogeneous content management processes: from web editing to collaboration (IsaWiki and WikiFactory), from e-learning (IsaLearning) to professional printing (IsaPress).

Contents

Chapter 1

Introduction

This work is positioned over two related research areas: markup languages and document engineering. Moving off an analysis of models of documents, their representation and division into constituents, the thesis proposes a new model based on *segmentation* and *patterns*. The basic idea is 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.

The thesis also proposes an actual markup language, IML, that captures the insights of the model. Later, it considers some publishing systems based on the model and IML, that have been implemented for e-learning, web publishing, collaboration, and professional printing. The point is that a radical simplification of markup practice facilitates the creation of a wide range of document tools that are inherently hard and highly useful.

The separation between content and formatting, as well as the need of segmenting document into subcomponents, is one of the most accepted (and flaunted) principles among document engineers and markup experts. It is not my goal to assess the importance of such an approach, whose advantages are undeniable and widely accepted. What I want to do is *extending in a radical way* that principle, and presenting some systems designed and implemented upon these ideas of extreme segmentation and normalization.

Then, the first part of my research addresses theories and techniques to model documents. The analysis and classification of markup languages deserves great attention, in particular the distinction between *prescriptive* and *descriptive* languages, and further subclasses of descriptive ones. Prescriptive languages are meant to prescribe rules that all documents must follow and are primarily used to label new documents, while descriptive ones are meant to describe structures that already exist and are primarily used to encode legacy material. Large space is devoted to clarify such distinction and to make clear where a generic approach is valid, and where more exhaustive, fine-grained and complex specifications are needed.

The point is understanding which are the most important features of a well-engineered document, which of them should (and can) be extracted, which can be neglected and under which circumstances. Then, I investigate the most discussed and thorny issues in that field: the separation between content and presentation (reporting also some opposite opinions), the conflict between hierarchical and plain documents and the importance of nested and unambiguous structures. Such a theoretical discussion comes alongside a description of tools and techniques to actually extract those relevant information (with particular attention to the WWW context) and reflow them in different documents.

I then propose a model, called Pentaformat, that refines the classical *content/presentation* distinction by identifying five dimensions of a document, able to capture and separate all its subcomponents. What is usually denoted as content is further divided in *content* and *structure*, in

order to indicate the plain text (and images) and the logical structures built on such a bare level. Those dimensions are clearly distinguished and discussed as two sides of the same medal, since they have different goals and roots. The (inter)connection between *structure* and *presentation* is further discussed and refined. Presentation is not "useless" but a powerful means to make more understandable and appealing for humans an information that already exists, i.e. the structured content.

Two more dimensions complete the model: *metadata* and *behaviour*. Metadata are all those information about a document that allow authors, managers and readers to make sense of its content in relation to other documents of the same kind, other documents related to this one, other versions or variants, other external resources. The "behaviour" dimension describes all the dynamical actions and events in a document. The increasing importance of interactivity and dynamic content is testified, for instance, by the last trend of the World Wide Web, where javascript applications, advanced toolbars, DHTML pages are being more and more successful. Actually metadata and behaviour are briefly investigated in this work, as an essential piece of the global vision of the team I belong to, but they are out of the scope of my thesis.

In fact, my focus is primarily on structured content and logical organization of a document. The problem I see is that a strong separation between content, structure and presentation is not enough to produce well-engineered and manageable documents. The risk is overdesigning structures and providing authors too many constructs and composition rules, although they are devoted to describe only raw information, without any presentation. The paradigm of separation between content and presentation is universally accepted in the community, and many markup languages have been proposed according to that rule (descriptive languages such as TEI, DocBook, XHTML, etc.). However, looking at their specifications some complexity is still evident and, in some cases, a sort of redundancy exists. It is quite common to find very complex documents, whose structures are difficult to be read and extracted. Yet, many times that complexity is intrinsic in documents' nature, or in their subject, but it is as much common to find documents that could have been written in a simpler and clearer way.

In my mind an adequate solution can be describing "best practices", or better identifying the most common and useful "patterns" that authors really need in their documents. Then, in the central part of this work I discuss some examples of overdesigned elements definitions, transformed into pattern-based definitions through an incremental process of simplification. Those examples lead us in discovering the seven patterns discussed here: *marker* (an empty element, whose meaning is strictly dependent on its position), *atom* (a unit of unstructured information), *block and inline* (a block of text mixed with unordered and repeatable inline elements that, in turn, have the same content model), *record* (a set of optional, heterogeneous and non-repeatable elements), *container* (a sequence of heterogeneous, unordered, optional and repeatable elements), *table* (a sequence of homogeneous elements) and *additive and subtractive contexts* (descendant elements added or prohibited in a content model, in any position).

My work steps a bit forward: rather than limiting to identify and investigate some useful patterns for descriptive documents, I suggest to only use exclusively them. My conclusion is that *any* document can be *projected* into a strict composition of a (very) limited set of objects, according to a (very) limited set of rules. The key aspect is that such normalized document expresses the same fundamental information of the original one, in a simple, clear and unambiguous way.

I foresee two possible applications of this approach: as a *constructive model* adopted by designers who want to create new and well-engineered resources from scratch, or as a *segmentation model* adopted by designers who need to extract information from legacy documents (and build applications that manage that information).

Adopting the minimalist paradigm embodied by patterns, I propose an actual markup language, called IML. The acronym IML mirrors both the role and the origin of that language: IML stands for "IsaWiki Markup Language" to remind the first system where it was studied and applied, but also stands for "Intermediate Markup Language" to indicate its capability of intermediate language for multi-channel publishing. It has a twofold goal: separating the actual content and the presentational aspects of a document, and expressing that content in a clear, unambiguous and well-engineered way. The idea, in fact, is that any document can be normalized

into IML, regardless of its actual layout and formatting. A full chapter describes this language, by clarifying its origin and applicability, in particular by comparing it with the existing markup languages.

IML has been successfully used as internal format for many applications, that will be described with this thesis. The core of my work, in fact, is that a radical simplification of markup practice facilitates the creation of a wide range of document tools that are inherently hard and highly useful. Actually these tools are the result of collaboration among different people and different development teams, but a central role has been played by the language (and, in general, by the pattern-based approach) proposed here. Some of them will be presented in the last chapters: IsaWiki, IsaLearning, IsaPress and, briefly, WikiFactory.

IsaWiki is a distributed publishing environment, aiming at realizing the *Writable Web* and the *Global Editability* paradigms. The term *writable web* indicates the possibility of transforming the World Wide Web into a platform where users can write pages, with the same skills and tools used to read them; moreover, *global editability* means that users can modify any web document, regardless of its location, access permission and data format. IsaWiki achieves (at least, partially) both these goals, without revolutionizing the architecture and protocols of the current WWW: it is a distributed architecture composed by a server that supplies services for registered users, and allows them to store customized versions of any web document, and a client-side editor/sidebar that monitors users' navigation and allows them to edit pages directly within a browser (through a WYSIWYG interface). The system stresses on content and layout separation: the idea is that authors are primarily interested in changing (and customizing) raw structured content, instead of whole pages. So, whenever an user asks to edit a web page editing facilities are activated only on that plain content. The document can be then converted and displayed in many other data formats. It can be again downloaded, modified and again uploaded onto the system, to be further converted. Such a complete independence of reading and writing from the actual documents' data formats is possible thanks to the IML normalization.

A similar approach has been adopted for the design and implementation of IsaLearning and IsaPress, two authoring systems respectively used for e-learning and professional printing. Although characterized in different ways, both these scenarios suffer a common problem: the high-quality of the final product very often implies a very high complexity in the authoring process, and requires the (manual) intervention of expert users. In both cases, what commonly happens is that authors write plain text and some experts transform it into products that respect all the domain-specific requirements. In case of e-learning they are experts of LCMSs (learning content management systems) that package content into reusable learning objects, while in case of printing they are make-up experts that paginate documents, fixing imperfections and variations.

A different solution consists of allowing authors to use their preferred productivity tools and extracting the relevant information they provide. IML is a natural candidate to address such issue. What IML does is representing in a structured and simplified way, an information that authors have previously written according to their preferences, (good and bad) habits, time resources and so on. By exploiting IML normalization, it is possible to completely automate the production of final high-quality results, from original sources. Then, authors can keep on working on those files and produce very good output, without having to learn new technologies and tools.

IsaLearning and IsaPress have been both designed on such a model: IsaLearning has been used as internal conversion tool for the project A^3 (in Italian, "Ambiente Accessibile di Apprendimento"), an open-source platform used to supply e-learning content and services at the University of Bologna. IsaPress is a conversion engine used by an Italian publishing house, called "Il Mulino", to automatically produce high-quality books from raw files. A paper version of the A^3 material has been created by using IsaPress as well. As I said, other software components are involved in the production process of IsaPress and IsaLearning (and many people participated to their implementation): what is relevant for this thesis, however, is the normalization of the input files in IML and the automatic extraction of structured content.

Finally, it is worth spending some words about one project I have been recently working on: WikiFactory. The rationale behind the project is the possibility of mixing advantages of free edit-

ing embodied by wikis, with advantages of semantic web technologies. Then, we designed an application that takes in input an ontological description of a domain, and automatically delivers a semantic wiki for that domain. Rather than on ontological aspects, I am working on the possibility of delivering the same content on different wiki platforms. That is possible by using a common language that captures all relevant information of a wiki page, and specialized conversion engines that instantiate that information for specific wiki clones. I am a co-author of that language, called WIF (Wiki Interchange Format), which as expected follows all the patterns discussed here.

The rest of the thesis is structured as follows. Chapter 2 discusses related works and main issues in document engineering and markup languages. Chapter 3 introduces my segmentation model, moving off some case studies. Chapter 4 focuses on design patterns for digital documents and presents a segmentation/constructional approach based on those patterns. Chapter 5 introduces IML, and a flexible architecture based on it. Chapter 6 presents IsaWiki, while chapter 7 presents IsaLearning and IsaPress. Chapter 8 briefly describes WikiFactory. Final remarks and ideas for future work are in chapter 9.

Chapter 2

Document Engineering

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 thesis: *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 (actually, no content could exist without markup) and study different issues and approaches for text encoding.

In this chapter I discuss the most thorny and relevant issues in these areas, trying to outline which are the most important aspects of digital documents authors (and designers) have to deal with. In particular, I divided such analysis in two sections: first, I focus on *documents modeling*, whose goal is understanding how a document can be represented in digital form, and second on *document analysis*, whose goal is understanding how that representation can be automatically extracted from legacy resources.

1 Modeling documents

Although implicitly, authors address a lot of fundamental questions, while writing a document: "Which logical structures do I need? How to divide document subcomponents? How to highlight details and specific features? How to represent complex data?", and so on. When they write a *digital* document new issues need to be solved: "Which is the most suitable format? Which editor can I use? Which constructs fit my requirements?", and in particular "Which markup language do I need?".

1.1 Different objectives, different mark-up languages

The first step to understand the nature of a digital document is understanding the nature of the language it is written in, the objectives it was designed for, and the basic principles underpinning it. In the literature, many classifications were proposed, each useful to capture some specific features.

Coombs, DeRose, et al.[CRD87] classified markup in six categories still universally accepted nowadays: punctuational, presentational, procedural, descriptive, referential and meta markup. *Punctuational markup* consists of using a set of conventional marks to highlight basic syntactical information about a written text. Periods at the end of sentences, commas to organize text, white spaces are all example of this kind of markup which, how remarked by authors, existed before the advent of electronic documents. The same can be said about *presentational markup*, which consists of all those graphical information useful to make clear high-level features of a text: spaces between paragraphs, pagination, enumeration of lists or blocks, vertical spaces among elements and so on. Text-processing systems introduced new kinds of markup, exploited by applications to render content or perfom more advanced operations. *Procedural markup* consists of a sequence of commands that indicate procedures a specific application should follow, such as 'skip a line',

'draw a letter', 'collapse words', 'display a table' and so on. Such a markup is obviously device-dependent and strictly related to a specific layout and formatter. A more advanced solution is the *descriptive markup*, which consists of identifying the role and type of each text token. While procedural markup indicates how a text fragment has to be treated, a descriptive approach indicates what a text fragment is, and which class it belongs to. Advantages of such approach are evident: flexibility, easy creation, separation between content and presentation, portability, modularity and integrability in other systems. The authors completed their taxonomy by mentioning *referential markup*, which consists of declaring entities externally to the document and substituting those entities during the actual processing, and *metamarkup* which allows authors and designers to control the interpretation of declarative languages and to extend their vocabularies.

Descriptive markup became more and more popular with the advent of SGML and, later, XML. Goldfarb[Gol81] outlined benefits of that approach stressing on the two main properties of a descriptive language: generalization and rigorousness. Generalization means that a such a language does not restrict documents to a single application, a single formatting or a single publishing process; a document is marked-up once, in order to describe its structure and attributes, and all future processing can be implemented over that representation. Rigorousness means that content and structure are expressed in a rigorous and unambiguous way, so that advanced and reliable applications can be actually built. A huge amount of papers, books and discussions about SGML and XML have outlined the power, flexibility and applicability of descriptive languages. I do not want to discuss them in detail, though I cannot omit citing the "canonical" references to Goldfarb's SGML Handbook[Gol90], Sperberg-McQueen and Burnard Introductions to SGML[SMB97], and XML[SMB00].

Descriptive languages have been referred as "generic" or "declarative", to emphasize the fact they state something about a document, rather than indicating how to process it. In this work, I call them "declarative" in stead of "descriptive". The reason is that the term "descriptive" can be used for a more fine-grained and specific classification, between "descriptive" languages and "prescriptive" ones. Such a distinction has been often made with regard to document models. Quin[Qui96] investigated descriptive and prescriptive DTDs: a prescriptive DTD prescribes a set of rules which all matching documents must follow and may be primarily designed to create new material; a descriptive one describes structures that exist, rather than to force any particular structure, and may be primarily used to create an electronic version of material that already exists (of course, a descriptive model may also be used to create new documents).

Extending this dichotomy to markup, *descriptive* can be used to refer to markup that simply states some quality about each text fragment, without trying to impose any rule on how and where it should appear, while *prescriptive* markup, besides simply providing names for the labels to use in the markup, also imposes constraints and structural rules on the use and positioning of labels. The meaning of the term 'descriptive' for this thesis will be deeply discussed in section 1.

Piez[Pie01] introduced a new category of markup, the "exploratory/mimetic" one. This markup details those features of the text that are relevant to the encoder, without requiring to adapt the content structure to the schema language, nor having the schema language completely predict the document evolution. The key aspect is the relation between an instance of document and its model: a text is marked up primary and the model is an *ex post facto* expression of something that the markup "discovered" about that text. Explorative/mimetic languages are not meant to impose constraints or dictate rules about the structures of a document, but to simply describe document instances. In a sense, an explorative/mimetic document model does not exist, but it is derived from instances afterwards. For this reason, Piez used the adjectives "mimetic" to indicate that a marked-up document aims at imitating its source, and "exploratory" because it is adaptable to that source. On the other hand, as the same author admitted, a pure exploratory/mimetic language is difficult to be justified and used in practice and the author himself recognized as "eploratory/mimetic" a fictional language he called ProfML, composed by a set of conventions that could be used in an exploratory way.

Renear[Ren01] described two dimensions in markup languages, *domain* and *mood*, by proposing an interesting parallelism with terms from linguistics and speech-act theory. The mood indicates the 'tone' of a language and it can be classified as "indicative" (when the language de-

scribes something) or "imperative" (when the language impose something). The domain indicates whether a markup refers to logical structures of a document or presentational aspects, and it can be classified as either "logical" or "renditional". Either a language designed to describe the actual content of manuscripts or its final formatting could be imperative or indicative (although, as Renear remarked, a "indicative renditional" language seems to make little sense). Restricting to the logical domain, for example, an indicative element says that the tagged text fragment *is* a specific "object", intrinsically and independently from its mark-up; an imperative one says that the same fragment *has to be* modeled as that object. In a renditional domain, an imperative declaration says that an element has to be rendered in some way, while an indicative one says that it is intrinsically rendered in that way (as expected, no language can be placed in this category since the presentation is something added or forced on logical information).

Discussing Renear's position Piez provided a clear example of indicative and imperative moods, with a comparison between the TEI DTDs and the DTD used by the W3C to markup drafts and recommendation. While TEI mainly aims at faithfully represent legacy texts already printed and published, W3C specs describe something that still need to be created and maintained in that form. Although they are both declarative, or descriptive, TEI elements correspond to something already characterized in a specific way (for instance, an epigraph *is* an epigraph before being marked up with the `tei:epigraph` element), while W3C specs characterize content fragments fitting them in contructs provided by the schema.

Piez[Pie01] also made explicit a classification based on time processing, i.e., whether a markup looks backward (*restrospective languages*) or forward (*prospective languages*): a retrospective markup language is one that seeks to represent something that already exists, while a "prospective" language seeks to identify the documents constituent parts as a preliminary step to further processing. Renear's and Piez's classifications partially overlap, so that, as Piez himself noted, the 'prospective' property corresponds to Renear's imperative mood, and 'retrospective' to indicative. Note that both looking-forward and looking-backward languages are declarative. What changes is their perspective and objectives: while a retrospective languages declares document's content with respect to a legacy model, a prospective one declares document's content with respect to a new logical model, amenable for particular kinds of processing. "Prospective", indeed, does not mean "procedural". In a virtual spectrum based on time-processing Piez placed from the retrospective edge to the prospective first ProfML, very far TEI, then Docbook, up to SVG and XSL-FO.

Piez completed his analysis with a philosophical/rhetorical classification: the distinction between *proleptic* and *metaleptic* markup languages. In rhetoric, a proleptic is a trope in which an expression or figure of speech takes its meaning from something that it will appear later. A proleptic markup is any markup where the meaning of the tagging is intimately connected with the expectations for processing it. Note that such meaning is connected with the estimates for processing, rather than the processing itself. SVG and XSL-FO, for instance, cannot be considered proleptic, but prospective since they already carry a specific rendering and processing for each object. On the contrary, W3C specs are proleptic because they indicate how an object has to be treated, without knowing the actual effects of that treatment. A typical example of proleptic approach is using titles in the creation of new documents: in that case, authors are simply saying to the further process "name that object as a title, and whatever you do on titles, do it on that object as well"; they are not suggesting how to process a title (as XSL-FO does by specifying formatting properties of a text block). The difference between XHTML (Strict) and DHTML further explains hte distinction between proleptic and prospective markup: while XHTML (Strict) states which is the role of each object (though related to its future processing), DHTML anticipates some features of that object by using scripts or browser-specific capabilities.

On the contrary, a metalepsis is a trope in which the meaning of an expression is related to something already happened in the past. Then, a metaleptic markup works by saying something about the past, but in order to create new meaning out of it. The most evident example of that approach is TEI: on the one hand, features of legacy material are registered and structured according to their legacy model, on the other hand such information can be used to built new and advanced applications. For this reason, Piez defined metaleptic a "retrospective tagging for

prospective purposes".

A complete different (and admittedly simpler) classification has been proposed by Wilmott[Wil02]. Wilmott identified two main categories of markup languages, whether they have to be interpreted by humans or automatically processed by machines, *human-based* and *machine-based* languages, and emphasizes their similarities and differences. The author stressed on the different ways humans and computers read and understand information, and derived principles to design or adopt either one kind of language: computers most easily recognize data and markup when examining data byte-by-byte, by removing extraneous fillers, in "binary" encoding; on the other hand, humans are helped by contextual information, whitespaces and clear text. The author concluded that both classes of languages are useful: any markup language should indeed be designed bearing in mind what it will be really used for, with a clear division between computer-use and human-use requirements.

1.2 Format and content: should (and can) they be separated?

One of the most accepted principles in designing markup languages is the distinction between format and content. Such a principle is so embedded and well-accepted within the community, that providing a complete list of citations is practically impossible. We could say that any decent book about SGML, XML and markup encoding has to discuss, and actually discusses, that paradigm. However such a debate is very much older than markup languages and reflects the classical controversy between "form and matter", "what and how", "in and out" discussed among philosophers, artists, aesthetes, semiologists and so on.

In the markup community, the seminal paper by Coombs et al.[CRD87](beside proposing a classification of languages) outlined benefits of descriptive markup in terms of *maintainability*, *portability*, *cognitive demand* and *authoring enhancement*. Properly tagged source files, authors wrote, never require modifications or, better, they require really few changes: editing is simpler, files are protected from corruption and few experts can format a huge amount of data, since presentation is applied automatically and in a second phase. Moreover, well-tagged documents can be ported over different platforms since the actual meaning of a document is captured by descriptive tags and specific conversion can be performed by trivial programs: different typesetters, different devices, different applications can display the same content simply by converting it on-the-fly. The markup process itself is simplified, since authors need only to select appropriate labels for content elements and they can do that with little more than the normal linguistic processing. What authors called "descriptive markup" can be strightfowardly read as separation between content and formatting: what really counts is the actual role of text objects, rather than their final rendering or processing.

In their introductions to SGML[SMB97], and XML[SMB00], Sperberg-McQueen and Burnard highlighted advantages of content/format separation. Authors 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". A content analysis program might read and extract footnotes, while a formatting program might collect them at the end of each chapter, and so on. Similarly an annotated text with names of places and persons, might be used to create an electronic index, or a paper print, or a source for data miners and so on. The summarization of the same authors might be the best explanation: "XML focuses on the meaning of data, not its presentation".

These theories have been consolidated with the development of XML technologies, and with the same standards proposed by the W3C. The massive use of CSS or XSLT recommended by the consortium (as well by all XML experts), the proliferation of books, articles, interviews about such paradigm, the increasing importance of accessibility and multi-devices issues, the diffusion of applications and softwares that properly embody that philosophy have made the concept of "content/formatting separation" almost indissoluble from the concept of advanced publishing. No one can now deny benefits and diffusion of that approach, and thousands of final statements about XML content/format separation can be found in the literature: "XML focuses on the meaning of data, not its presentation" by Sperberg-McQueen and Burnard[SMB00], "the ability of XML is its ability to separate the user interface from the data" by Pardi[Par99], "XML markup describes

a document's structure and meaning. It does not describe the formatting of elements of the page" by Harold[Har99], or "XML helps us turn what is otherwise a stream of information into structured, manageable and meaningful data" by Laurent[Lau98] and a much more longer (practically infinite) list of similar citations.

Some "odd" but interesting parallelisms are also worth being cited. A very complete, and admittedly complex (at least for me, as a computer scientist) discussion was written by Liu[Liu04]. The author compared the separation of content and format in markup languages with some philosophical theories, and stated that data floods from transcendental sources toward actual documents, where their essence is presented in a human- or machine-readable format. Those transcendental sources are blind spots that might have even been called the Sublime, the God: users are compared to prayers who "query" the God, where the term "query" means they actually perform an SQL query or an XSLT statement to collect and re-organize data. Piez[Pie01] proposed an historical parallelism by comparing our XML era with the postindustrialism of the first 19th century. The logic of separating content from presentation reflects the principle of division of labor joined with the principle of making the component parts to be interchangeable, which made possible the "American System of Manufacture". When the individual components of a manufactured item were submitted to quality control mechanisms higher-order economies could be realized, and manufacture could be improved and speed up. Similarly the logical separation provided by XML and related technologies has bring users many advantages in terms of scalabily, portability and flexibility of digital documents.

Instead of further expatiating upon positive opinions, I prefer to investigate some "opposite" positions that contradict a so accepted and basic principle, but might help me to clarify the applicability of my approach. The same Piez[Pie05] expressed a very interesting poinf of view, though perfectly compressed in the title of the paper. Piez noticed a gap between what markup designers promised to achieve about separating content and format, and what they actually achieve. In particular, he argued that any schema based on that approach cannot be enough to model all scenarios and needs markup practitioners have to deal with: as far as complex and complete, a schema cannot forecast all its applications, so that there would ever be something that cannot be correctly modeled. For instance, he cited the markup used in the scientific conferences' proceedings, where authors do not have native constructs to markup poetry or verse, or markup for automatically generated indexes, where the actual relation among title subcomponents (and their inner structure) is very often hidden or masked. According to Piez, the problem does not rely on the incompleteness of those languages (that could be continuously enriched), rather on the practical impossibility of forecasting every feature and detail readers are interested in a text. The consequence is that also the descriptive schemas (such as TEI and DocBook) may trend to be "attracted" into applications semantics and the tagging process risks to be more oriented towards a particular outcome, rather than towards a pure descriptive tagging of text.

Piez claimed that some text carries something (almost) indescribable, which is very hard to be markup up. He resumed philosophical concepts of 'noumenon' and 'phenomaenon' ('noumenon' is 'the thing itself', the basic reality underlying a 'phenomenon', which is an observable event) to clarify his point: descriptive markup moves toward a noumenal dimension but never escapes a phenomenal one. I found really meaningful an example provided by the author about poetry encoding: while it is quite simple to format "transparently" a poem into an HTML page, it is very hard to design a markup language that could markup up everything users might be interested in (that could also be strictly related to 'format'). Author's conclusion was that "there is no reason to fear or disdain presentation-oriented design. We need only to discriminate when we want and need an isolated layer for our information capture, and when we want to work more directly with the 'hot lead' ". Upon those ideas, Piez proposed a markup language called WGLL (Web Graphics Layout Language) openly based on presentational aspects but used to mainly label textual documents. WGLL can be intermixed with SVG, reduces the effort in producing SVG files lightening the burden of tagging and encoding, and was (philosophically and practically) really close to SVG. What is relevant here, more than syntactical and applicative aspects of the actual language, is the *interpenetration* between content and format discussed by Piez, and the predominant role he gave to format.

Hillesund[Hil02] wrote another, though quite different, invective against the separation between content and format. He argued that the doctrine of "one input - many outputs" so flaunted within the XML community is basically wrong. On the contrary, being impossible to reflow a single content into different layouts, for different purposes and different media, the only possible paradigm is "many inputs - many outputs". His theories can be summarized in two points: *content/format interleaving* and *impossible reuse*. According to Hillesund, the separation between content and presentation is getting confused and misleading when applied to books or other publications. In particular, despite what all XML expert say, there is no way of separating those components but they are strictly interrelated and mutually dependent. Rather than being an extra layer built upon the content, presentation is an irreplaceable part of a document that expresses a kind of "semantic" information and affects the way users perceive and comprehend a text. Structures like titles, abstracts introductions, chapters are considered either semantic and typographic structures, since authors actually use typographical elements, when defining the logical structure of a document. The author observed that such a behaviour is rooted in the history of typography and have not changed since the birth of the paper printing itself. His conclusion is that XML technologies cannot expect to segment elements that are intrinsically intermixed and have always been living together.

The second point of Hillesund is the impossibility of reusing content fragments and merging them from different sources into a (good) composite one. In that case, the logical order of elements is distorted and the original information is scattered over a document probably unclear, incomplete or too complex. The metaphor used by the author is really explicative: reusing content can be compared to take a pair of scissors, deface a tapestry and rearrange pieces of cloth, in order to obtain a new and appealing tapestry where everything is well-connected and harmonious. The conclusion is that there is no easy way to manipulate fragments as users need and prefer: without actually re-editing content, author said, publishers cannot take single chapters of a paper book and transform them into on-line pages, cannot directly transfer legacy material into learning objects, cannot automatically deliver content on mobile devices from printed material, and so on. However, the focus here is clearly on the editorial interventions needed to adapt a text for re-purposing, rather than on the technical feasibility of that approach.

Walsh[Wal02] published a point-to-point response to Hillesund, in the same journal. Basically the author said that content/format separation is possible either from a logical or practical perspective and held DocBook as example of the success of such distinction. In particular, he focused on the feasibility of re-purposing content on different media, with different layout and different formatting. I found the position of Walsh very interesting since he stated that "a perfect separation of content and presentation is not always possible, but it is often possible to come very close"; in particular, he explained that it depends on how much the content is suitable to be extracted and reflowed. I completely agree with the author, when says those problems are mostly editorial in nature and cannot be solved by technical solutions. Stating *a priori* that separating content from presentation is impossible in practice, as Hillesund did, is very different from stating that a bad (intended as 'not designed for that purpose') text cannot be extracted, manipulated and re-formatted with perfect results.

1.3 Plain or hierarchical? or what else?

Separating content and format is only a partial step for creating well engineered documents. Another important point is deciding the overall structure of a document, in particular by choosing between a plain sequence of objects or a hierarchical structures of containers. Although the second solution seems to be better than the first one, many applications and users still produce documents with a non-hierarchical internal structure.

The discussion about hierarchical structures is rooted in the first years of markup languages development. An analytic and philosophical approach, the OCHO model, was discussed in the early 1980s by Coombs et al.[DDMR87]. According to OCHO 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 rela-

tionship among objects (for any two objects within a book one comes before the other), and it is made of plain units of information (content). Authors divided advantages of an OCHO-based approach in three main categories: composition assistance, production assistance and facilitation of alternate use of data. First of all, OCHO helps authors since they can deal with a document at an appropriate level of abstraction and, rather than concentrating on its formatting, can work on its logical structure and relations among elements. Writing, collaboration and composition 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 OCHO helps users in manipulating and understanding documents, that can be treated as databases of text elements: since relations and dependencies are explicit advanced retrieval functions can be implemented, as well as functions of content fragments composition and reflowing.

OCHO raised a great interest within the community and many commentaries and rensponses were published. Particularly interesting was the objection of Dicks[Dic97], related with the controversy between format and presentation. Basically Dicks said that OCHO had a low support for images and formatting properties, and downplayed the renditional aspects of a document. On the contrary, he believed that text and presentation are both integral and irreplaceable part of a document identity. DeRose[DeR97] answered saying that OCHO did not consider presentational features 'unimportant' rather *derivative*. 'Derivative' because they depend on the level of abstraction at issue: presentation actually effects the reader's recognition of content objects but it derives from the medium used to access content. While Dicks basically agreed with the OCHO hierarchical model, Bringsjord[Bri96] proposed a completely different organization according to which a document is a collection of unadorned chuncks of information (jottings) and some procedures useful to manage and display them. Bringsjord's position can be defined 'bottom-up': instead of being pre-defined and hierarchicual structures, which map existing conceptual organizations, documents derive from the assembly of 'pure' jottings, moved and merged according to the user's need and preferences.

Actually few counterproposals to OCHO were done (and they had a very low success), and OCHO suddenly became the most adopted model for designing markup languages. However, the OCHO philosophical approach had been preceded by the same SGML (its ancestor GML). In the following years SGML would be followed by XML, as further evidence of the flexibility and power of a hierarchical model. Although SGML and XML community has widely discussed benefits of a hierarchical organization, many non-hierarchical documents can be found today, many software produce plain content and, above all, many authors use to write documents without an explicit hierarchical organization of subcomponents. The example of XHTML is meaningful. XHTML headings (`h1`,..., `h6`) are meant to implicitly indicate the logical organization of a page, although they are in sequence. The following fragment shows an example of a page with three levels of headings, written in XHTML 1.0 and XHTML 2.0. The latter introduces tags `h` and `section` to make explicit the nesting relations between elements. The introduction of those tags shows how much the importance (and need) of hierarchies and structured content is shared by markup researchers.

```
<body>
        <h1>Title 1</h1>
        <h2>Title 1.1</h2>
        <h1>Title 2</h1>
        <h1>Title 3</h1>
        <h2>Title 3.1</h2>
        <h3>Title 3.1.1</h3>
</body>
```

```
<body>
        <section>
```

```
        <h>Title 1</h>
                <section>
                <h>Title 1.1</h>
                </section>
        </section>
        <section>
        <h>Title 2</h>
        </section>
        <section>
        <h>Title 3</h>
                <section>
                <h>Title 3.1</h>
                        <section>
                        <h>Title 3.1.1</h>
                        </section>
                </section>
        </section>
</body>
```

A plain document is certainly simpler to be created (actually it is what most word processors produce), less verbose and matches the visual structure of the document itself. On the other hand, deriving its actual structure requires further operations, since both humans and applications have to count objects and recollect them on-the-fly. Despite the model it adopts, a document like the example is intrinsically organized in nested containers and, at the end, paragraphs. Only a methodical use of hierarchies allows users to express those logical relations and to avoid ambiguity in their interpretation.

For this reason, tree structures have raised great interest in the database community as well. Glushko and McGrath[GM02] discussed how wrappers and containers make clear the functional dependencies among elements. He noticed that documents are traditionally studied from two perspectives: *document-centric* analysis, which considers them as artifacts that are perceived as a rendition (combination) of presentation and format; and *data-centric* analysis, which considers them as artifacts more regular in their structures, with a minimal presentation and more consumable by applications than by humans. The author argued that these approaches have a lot of unexpected common aspects and can be managed together in order to obtain well-engineered documents.

Particularly interesting is the parallelism between the process of database normalization and the use of (sub-)structures in the documents to express dependencies and nesting. Date[Dat81] stated that "recognizing functional dependencies is an essential part of understanding the meaning or semantics of the data". Glushko said (and I completely agree) that relational databases use normalized tables to express hierarchies, as well as wrappers and containers give structure and depth to the documents. Well-structured information in fact allows users to minimize redundancy, localize dependencies and ensure that information reflects the features and constraints of the application domain.

Salminin and Tompa[ST01] described documents as assemblies of structures and components based on a required business context. The "document assembly" is the process of constructing a new document instance from those basic units. Such process is simplified and empowered when documents adopt containers. They play a role similar to the database normalization: as normalization increases efficiency and prevents anomaly operations, as wrappers facilitates documents' updates and transformations. The point is that relations among subtrees and branches are clearly expressed and easily (and reliably) modifiable in a tree-based document.

But modeling documents as trees is not enough. Very often users need to markup structures that do not not nest neatly into others, and cannot be represented as a hierarchical structure with a single root. A classical example is a text with a quotation starting in the middle of one paragraph and ending in the middle of the next: by only adopting an OCHO model, it is impossible to markup either paragraphs and citations in that scenario. Different techniques have been pro-

posed by researchers to handle multiple and overlapped hierarchies.

An initial recognition of the problem, although it did not define an analysis and processing model in sufficient detail, was the SGML CONCUR. It is an optional feature of SGML usable to annotate concurrent hierarchical structures in a single document. Basically, CONCUR allows authors to use different DTDs in the same schema, by assigning elements to the DTD they belong to by preceding them with the corresponding document type (surrounded by round brackets). Compatible with SGML, CONCUR cannot be used for XML documents and proved to be very complex and inefficient in handling self-overlap and relations among DTDs.

The same authors of OCHO revised their theories to accomodate overlapping, by proposing OCHO-2. Renear et al.[RDM96] says that different hierarchies of content objects can be found in the same text, depending on different analytical perspectives, but they are never applied simultaneously. For instance, prosodic objects (stanzas, lines, half lines) do not overlap with each other, nor do the linguistic objects (sentences, phrases), nor do the dramatic objects (acts, scenes, casts). Although different hierarchies actually exist on the same document, they do not interfere with the most common encoding practices. Practical counterexamples brought the same authors to propose OCHO-3, according to which each analytical perspective can be further divided in sub-perspectives and for each of them simultaneous hierachies can be identified. A sub-perspective is a discipline or any other unified part of an analytical perspective: authors proposed the example of'literary' studies, which can be divided in 'literary history', 'literary criticism' and 'textual criticism', which in turn can be splitted in 'transcription', 'recension' and 'emendation'. At that level, multiple hierarchies are distinguished and they overlap moving up to the higher levels of abstraction.

Sperberg-McQueen and Burnard [SMB02] introduced a different method based on segmentation, consisting of marking up the original document as smaller fragments that do not overlap and compound them into multiple hierarchies. Although very powerful, such method makes difficult managing complex units of content (and re-building implicit structures) and have had a low acceptance within the community. "Join" techniques were also proposed in the same guidelines to augment the power and the applicability of such segmentation approach.

Durusau and O'Donnell[DO02] proposed JITT, a solution which does not use XML syntax (even if it is very close) but provides users a powerful framework to handle overlapping elements. Basically, it relies on specialized "filters" that count only some tags of a document, ignoring others. The key aspect is that JITTs markup is recognized at processing-time rather than encoding-time: documents can be bad-formed when edited and saved, but can be transformed into processable XML while parsing them. In a sense, JITTs allows users to express different coexisting views on the same document, and to select them on-the-fly whenever (and by whoever) those documents are accessed. A similar proposal in non-XML syntax is LMNL(Layered Markup and Annotation Language) by Tennison and Piez[TP02]. LMNL is a markup notation that reflects an abstract model, where documents are strings over which span a number of named ranges and metaranges, and can be easily translated into a single view of source data, for instance an XML tree. DeRose[DeR04] proposed CLIX (Canonical LML in XML), an XML syntax for LMNL: any LMNL document is normalized into a CLIX one that can be processed and validated by XML tools. CLIX development is not finished yet: for instance Bauman[Bau05] recently proposed some refinements.

Sperberg-McQueen and Burnard[SMB02] proposed a quite different solution named "Milestones": replacing nested troublesome elements with empty elements that indicate the start-point and the end-point of the fragment which overlaps. Attributes and crossing references can be then used to express relations and dependencies between those anchors. Milestones are quite difficult to be handled (since hierachies are hidden and sometimes hard to be extracted) but they can be processed by XML tools and have a very low inteference with the original text.

However, the less invasive technique to handle overlapping hierarchies is the so-called "standoff" markup. It consists of expressing markup in a different location from the content it applies to, and later applying it through a process of "internalization". Standoff markup is really powerful and flexible, but very difficult to be handled since conflicts, changes, moves of fragments influence (and can break) addressing mechanisms. Different W3C standards can be exploited

to implement external markup: XPointer[GMMW03] to express complex locations, ranges and addresses within an XML document, XLink[DMD01] to create complex links including external ones (which can be used to state something about a text fragment retrieved with an XPointer) or XInclude[MO04] to describe content fragments and their inclusions and reuses.

2 Analyzing documents

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, retrieved and transmitted. However, it is very common to have paper documents, whose digital sources are unavailable or corrupted, that users need to transform into electronic and more structured resources. Moreover, in the field of World Wide Web automatic extraction of structural information is increasingly becoming more and more important. For this reason, a lot of tools can be found in the literature about *a posteriori* analyses and re-structuring of both paper-based documents and web pages. An exhaustively discussion is not in order here, but an overview of those techniques and, in particular, a discussion about the models of documents they adopt can be really useful for the purpose of this work.

2.1 Structural analysis of paper-based documents

A lot of techniques and systems were proposed by researchers and professionals in order to automatically convert paper-based documents into electronic ones (Song Mao et al.[SMRK03] presented a very interesting survey). Basically, they all try to rebuild two electronic representations of a document: the *physical layout* and, then, to map it into a *logical structure* of content. 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. Nagy et al.[NSV92] adopted that approach: the system Gobbledoc performs a syntactical analysis to divide a document into labeled rectangular segments. Then, it performs a string-based research over their content and allows users to search, navigate and display those fragments. Two complementary views of the same document are provided to the users: a set of images that report the original graphical information, and a set of text segments that report the informative content of the original source. Bottom-up solutions start with single pixels and cluster them into letters, then into words and paragraphs, then into graphical areas up to rebuild the whole document. Anil et al.[JY98] presented an iterative approach for region identification, that exploits the connectivity and contiguity of graphical elements in order to extract text fragments, tables, images and drawings. Although, such approach primarily addressed technical journals, authors used and evaluated similar techniques with other documents too. Similarly O'Gorman[O'G93] introduced a method to extract layout information on the basis of nearest-neighbor relations. In particular, the system evaluates spaces between elements, text orientation, skews and areas borders and derives a logical representation of documents, as sequences of text blocks and lines. The system is highly configurable and proved to be reliable over different kinds of paper documents.

A transversal classification is very common in this field: *rule-based* versus *grammar-based* systems. Rule-based systems exploit transformation rules to extract logical structures from layout information, while grammar-based systems model the relations and nesting among document's elements with formal grammar production rules.

Niyogi and Srihari[NS95] presented Delos, a rule-based control system based on a multi-level knowledge base. The system implements an hybrid approach: text blocks are built with a bottom-up approach and then classified by analyzing their location and features. Later, three set of rules (basic knowledge, control and strategy rules) stored in a shared knowledge-base are applied to derive the logical tree structure of the original documents. Some years before Fisher[Fis91] had presented a similar approach that produced a logical representation of a document, in a format called MIF (Maker Interchange Format), by applying three set of extraction rules: location clues (about the position of graphical elements), format clues (about the presentation of content blocks)

and textual clues (about the actual content of each block). More recently, Lee et al.[LCC03] presented a geometric approach able to extract the logical organization of hierarchical documents and express it in a SGML/XML syntax. In particular, a knowledge base of geometrical characteristics and spatial relations is used as basis for automatic splitting and grouping of regions into layout subcomponents.

Grammar-based methods are widely used as well. Brugger et al.[BZI97] used a statistical document model to describe relations among layout entities, based on *n-grams*. In particular, the system models the most relevant patterns that exist in trees representing the logical structure of documents, and labels the input document with probabilities for each of them. By applying statistical inferences and incremental learning algorithms over the abstract model, the system automatically extracts the logical tree that maximizes those probabilities and minimizes recognition errors. Krishnamoorthy et al.[KNSV93] proposed a two-phases algorithm to extract logical structures from technical documents. The algorithm separately applies grammatical inference to study the horizontal and vertical projections of the input document. Each of them is segmented by a bottom-up approach which clusters lines in text blocks, text blocks in areas, and areas in super-areas (actually authors uses a parallelism with atoms and molecules). Finally, those two views are merged into a linearized tree, and further processed to obtain the final result.

Analysis systems can be also classified for the scope and generalization of their approach. Most of them, in fact, apply heuristics and assumptions suitable for specific domains but less powerful on different kinds of documents: for instance, Tsujimoto and Asada[TA90] proposed a system to handle multi-article documents such as newspaper, or Srihari et al.[SYG99] proposed a tool to interpret (unordered or incomplete) postal addresses, or Kim et al.[KLT03] an application to extract bibliographic records from a medicine databases, and many more examples could be cited, each used by a specific community in a specific context. On the other hand, many general-purpose systems have been proposed as well. Klink et al.[KDK00] proposed a rule-based system where rules are clearly distinguished in two classes: generic rules suitable for each domain, and specific ones that users can customize and detail for their own purposes. Aiello et al.[AMTW02] presented a more complete system to analyze a wide range of documents. The system is a monolithic application where different techniques are mixed together: statistical decision trees, geometrical information, annotations by users, coniguity and elements order, spatial relations are all used to derive both the layout and and the logical structure of the input document.

2.2 Structural analysis of web pages

The automatic analysis and segmentation of documents has increasingly been gaining importance in the field of WWW, as well. Although, the HTML language was born to provide a simple and easy-to-learn markup language for physicists to write their papers and publish them on the Internet, very soon its success spread over individuals, companies and oganizations. And needs and specifications became more and more complex.

By means of HTML tricks, tag exploitation, and creative use of graphics, page designers have managed to make a boring structural markup language become the means for incredibly complex and sophisticated interactive events available through web browsers. Of course, the drawback of this evolution was that most professionally created web pages started to include large markup sections aimed at decorative and layout purposes, and that these are often intermingled with the actual content of the page: although human readers can in most cases easily tell apart content and presentation, machine interpretation of the content is seriously hindered.

Applications that require to identify and classify subparts of web pages are high, and their aims are as wide and numerous as the applications themselves. The techniques they implement vary considerably. Kovacevi et al.[KMMV02] used a hierarchical representation of the screen coordinates of each page element in order to determine the most common areas in the page, such as header, side menu (either left or right), main content area, and footer. This analysis exploits the expected structural similarities between professionally designed pages as suggested by usability manuals and implemented by competitors.

Mukherjee et al.[MYTR03], Nanno et al.[NSO03] and Yang and Zhang[YZ01] all propose a semantic analysis of the structure of the HTML page, aiming at the discovery of visual similarities

in the contained objects in analogous pages. The fundamental observation is that the standardization in the generation tools of web pages has created consistencies in the style of headings, records and text blocks of the same category. Unfortunately, there are many ways in HTML to obtain the same effect in terms of font, color and style (tag names, order of tag, use of inline, internal or external CSS styles, etc.). So clearly similarities in the final effect may well correspond to differences in the underlying code, which adds a further layer of complexity in the process. Also Chen et al.[CZS+01] propose a method for classification of elements in a web page based on their presentation, nature and richness in style information.

Chung et al.[CGS02] proposes a restructuring approach to derive properly nested XML documents from HTML pages, by exploiting information about how HTML is typically used to render information, and how such visual markup is related with the logical structure of a document. In particular, authors consider a document as a hierarchical structure of block level objects (in terms of headings, paragraphs, lists, tables) and in-line level objects (bolds, italics, spans, etc.), where objects of higher level of abstraction are described by objects of finer level of abstraction. The meaning of a node is not directly associated with the object itself but it is embedded in the content and context of that node. Then, they propose a three-steps and bottom-up process to restructure a DOM tree: analyzing text in order to identify atomic units of content, grouping those units in more complex structures and polishing those structures by removing non-relevant or temporary information.

An automatic mapping between visual and logical information of HTML pages has recently been proposed by Burget[Bur05]. Rather than working directly on the HTML code of a page, the author propose an abstract model to describe visual appearance of a page. The interpretation of some features in that visual model, produce a description of the logical structure. Basically, a page layout is modeled as a graph of areas (edges between two areas means they are nested or contiguous) and weighted strings elements for attributes and text. A logical tree is extracted from the nesting of elements and the weight of their subcomponent. Then, specific information are gathered and expressed in XML, by identifying subtree and paths and by inferring tag names with a content-based analysis.

The automatic analysis of web pages has been widely studied in the field of *web wrappers* too. Wrappers are applications that extract specific information, elements and subparts of a page and present that information in a clear, unambiguous and processable way. They are primarily used to automate navigation, to improve efficiency, to deliver content on diverse devices or to re-organize structured data. Two main approaches exist for wrappers extraction: grammatical inference and inductive learning. Hong and Clark[HC01] uses an inference alghorithm to derive a grammar that describes the logical structure of a document, in terms of production rules, from a set of examples. Such grammar is later exploited by a wrapper that automatically extracts information from other related and similar pages, and uses them as input for data querying and mining. Crescenzi and Mecca[CM04] proposed an abstract model to capture the most relevant structures used in HTML and express them in a formal language, called *prefix mark-up language*, upon which information can be inferred without errors. Authors provided a theoretical discussion of such properties of validity and correctness and presented their grammar-based inference alghorithm.

Examples of inductive learning approaches are alike common in the literature: Freitag[Fre98] presented SRV, a general framework for content extraction based on a training phase when the system acquires lexical and linguistic information, and an actual analysis phase when those information are used to extract informative content. While SRV is mainly focused on free-text many solutions have been proposed to analyze semi-structured data: for instance, Knoblock et al.[KLMM03] introduced a wrapper algorithm that learns extraction rules from data inserted by users, through a graphical interface. Users initially mark-up page fragments indicating their semantic role; the algorithm subsequently exploits those indications to process more pages and extract relevant information. Within those applications, a very important role is played by NLP (Natural Language Processing) techniques, as proposed by Deriviere et al.[DHN06]. Those techniques share common objectives with my thesis: extracting logical information from plain text. On the other hand, they apply a bottom-up approach and try to build new structures from un-

structured content; my focus is primarily on translating and normalizing structures already (and explicitly) created by the authors.

A different application of web content analysis is the repurposing of content for PDA or small devices. Chen et al.[CMZ03] obtained such reflowing by determining the criteria for identify the elements of the page that constitute a content unit. The process is iterative, starting from a single block as wide as the whole page, and then progressively determining sub-areas within the areas already determined. Gupta et al.[GKDG03] performed the reformatting of the web content for smaller PDA screens through the filtering of specific nodes of the DOM tree and leaving only relevant nodes. In particular, authors proposed two set of filters: basic removers, which quickly remove some selectable tags and attributes, and advanced filters, which change or re-organize page objects such as advertisements, link lists, empty tables and so on. A simple graphical user interface can be used by the system administrators, to active and deactivate those filters in a proxy-based Java application. Penn et al.[PHLM01] proposed a different set of heuristics to extract tabular data and link from news web sites, and presented their prototype and experimental results.

Buyukkokten et al.[BKGM+02] proposed a very interesting "accordion" model for web pages, exploited to automatically extract and reflow information. A document is considered as a hierarchy of individual content units called Semantic Textual Units (STUs). Those units are built upon syntactic features of HTML documents, then organized according to the role of each of them. The relevant aspect is that hierarchies do not reflect the structural tree organization of a page, but the level of importance of each STU (for instance, H1 and B elements have the same relevance, higher than a normal P). Moreover summarization is performed over each single unit and readers can select which (and which version of each) unit they want to display in their PDAs. Such approach can be then considered a sort of re-structuring of web content and implicitly relies on a strong segmentation model, as the one presented with this work. Kaasinen et al.[KAK+00] presented a very similar approach, which considers a web page as divided into units compared to "cards in a deck". The system splits content into small units presented once a time to the users, who can select only one of them. The tree-based structure of an HTML document is then flatted into a sequence of cards, each with a specific level of relevance used to decide their initial surfing order.

Many more systems could have been listed in this section. A complete overview is out of the scope of this work, which indeed shares several aspects with these applications: first of all, the automatic and *a posteriori* interpretation and reconstruction of (sometimes bad-structured) content.

Chapter 3

Document Segmentation

In this chapter I propose a new model to segment documents, called Pentaformat, able to identify their most relevant constituents in order to process, recombine, and repurpose them for different domains and applications.

Defining what a document really is and what it is used for, is a complex issue widely discussed among different communities, over the years. Heterogeneous objectives as well as heteregeneous skills, interests and backgrounds lead scholars to stress on some aspects more than others: for instance, semiologists focus on languages and signs, computer scientists on automatic analysis and transformations, communication experts on message passing and immediacy, antropologists on users' reaction and so on.

A basic concept relies behind all these interpretations: a document is the result of a writing process, intended to store and communicate information. It's no accident that the word roots 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 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 by specific roles, in a specific time and for specific goals. The Pentaformat model stresses on such idea of documents segmented into distinguished but connected constituents.

1 What is text, really?

Although a vision centered on reusable components is increasingly gaining relevance for digital publishing, it is not a prerogative of the only last generation documents. The same basic constituents can be identified in any kind of document: either in old manuscripts or modern e-books, as well as in advertisements or recipes books and so on. What change are the actual elements in each cathegory, their properties, their final rendering but the logical segmentation keeps beeing valid.

1.1 Segmenting a manuscript

Consider, for example, the manuscript shown in figure 1. This is the original copy of the treatment "Of Colors", about optics, written by Isaac Newton and included in his most complete laboratory notebook (the whole notebook is found today in the Cambridge University Library, as a bound volume of over one hundred-seventy folios). Actually, only the first paragraphs are displayed from the original folio.

Newton had organized the text by following rules and stylistic conventions. He chose the most suitable constructs for expressing and structuring what he had in mind: so he divided the content in paragraphs, numbered them and highligthed some fragments. The writing and organising of a text, in fact, is done according to more or less predefined conventional patterns or

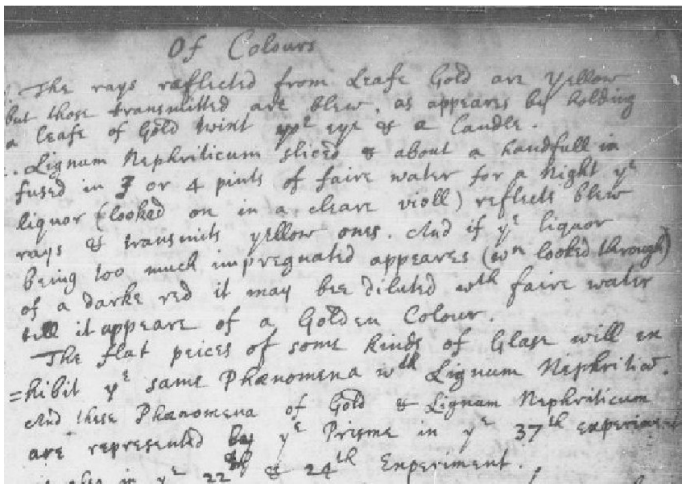
Of Colours
The rays reflected from Leafe Gold are Yellow
but those transmitted are blew, as appears by holding
a Leafe of Gold twixt ye eye & a Candle.
Lignum Nephriticum sliced & about a handfull in
fusd in 3 or 4 pints of faire water for a Night ye
liquor (looked on in a cleare violl) reflects blew
rays & transmits yellow ones. And if ye liquor
being too much impregnated appeares (when looked through)
of a darke red it may bee diluted with faire water
till it appeare of a Golden Colour.
The flat peices of some kinds of Glase will ex
=hibit ye same Phaenomena with Lignum Nephriticum.
And these Phaenomena of Gold & Lignum Nephriticum
are represented by ye Prisme in ye 37th experiment
& 24th Exprirment.

Figure 1. A fragment of the treatment "Of Colors" by Isaac Newton

genre norms which have been developed to fulfil specific communicative objective, as observed by Halliday and Hasan[HH89]. Djik[Dji80] divided these norms in three levels: at macro level, they define the overall structure of a text and generic elements like sections, subsections, images, notes and so on; at middle level, they define specific elements needed to write a poem instead of a book, to dramatize instead of telling a joke and so on; at micro-level they define all those rules, specific terms, words and features needed in a specific context. Regardless of the level we are interested on, it is evident the presence (and, meanwhile, the need) of 'something' that gives a structure to the raw content and makes it readable and consistent with the expectations of the readers.

Moreover Newton wrote those notes with his handwriting, with specific font, colors and on a specific kind paper (or whatever else). Time after, scientists and students would have transformed the original text (or handed on versions of it) into different pages, probably trimmed with comments, notes and so on. In other words, both the authors and his readers/publishers add a new layer to the structured content, a way to better show the document: the presentation.

Thousands of digital copies of this work exist today as web pages, e-books articles, XML files, PDFs, etc. Consider, for instance, the following TEI fragment that encodes the manuscript:

```
<tei>
...
<title>Of Colours</title>

<p id="3975_p1">
1. The rays reflected from Leafe Gold are yellow but those transmitted are blew, as
appeares by holding a leafe of Gold twixt your eye &amp; a Candle.</p>

<p id="3975_p2">
2. Lignum Nephriticum sliced &amp; about a handfull infused in 3 or 4 pints of faire water
for a Night the liquor (looked on in a cleare violl) reflects blew rays &amp; transmits
yellow ones. And if the liquor being too much impregnated appeares (when looked through)
of a darke red it may bee diluted with faire water till it appeare of a Golden Colour.</p>

<p id="3975_p3">
3 The flat peices of some kinds of Glase will exhibitthe same Phaenomenawith Lignum
Nephriticum. And these Phaenomena of Gold &amp; Lignum Nephriticum are represented by the
Prisme in the 37<super>th</super> experiment as also in the 22<super>th</super>  &amp;
24<super>th</super> Experiment.</p>
...
</tei>
```

The encoder has divided the text into blocks corresponding to the paragraps used by the author, with specialized elements for special blocks (as the title) or inline fragments. Those constructs reflect the organization of plain text done by Newton and, indirectly, the abovementioned

rules and conventions in structuring scientific texts. Macro-structures are evident, so that further tranformations into HTML pages or XSL-FO files, and simple text analyses, are easy to be implemented. However, the same content could be encoded in a completely different way:

```
<tei>
...
<title>Of Colours</title>

<lg id="3975_p1">
<l>1. The rays reflected from Leafe Gold are yellow</l>
<l>but those transmitted are blew, as appeares by holding</l>
<l>a leafe of Gold twixt your eye &amp; a Candle.</l>
</lg>

<lg id="3975_p2">
<l>2. Lignum Nephriticum sliced &amp; about a handfull </l>
<l>infused in 3 or 4 pints of faire water for a Night the</l>
<l>liquor (looked on in a cleare violl) reflects blew</l>
<l>rays &amp; transmits yellow ones. And if the liquor</l>
<l>being too much impregnated appeares (when looked through)</l>
<l>of a darke red it may bee diluted with faire water</l>
<l>till it appeare of a Golden Colour.</l>
</lg>

<lg id="3975_p3">
<l>3 The flat peices of some kinds of Glase will </l>
<l>exhibitthe same Phaenomenawith Lignum Nephriticum.</l>
<l>And these Phaenomena of Gold &amp; Lignum Nephriticum</l>
<l>are represented by the Prisme in the 37<super>th</super> experiment</l>
<l>as also in the 22<super>th</super>  &amp; 24<super>th</super> Experiment.</l>
</lg>
...
</tei>
```

The labels are used to mark lines of the manuscript: instead of emphasizing macro structures, the encoder re-built the 'physical' organization of the original document in order to track minimal details of the original copy hand-written by Newton. A point is worth being remarked: this encoding is not an artificial and not meaningful re-reading, but a different organization of the same content already living in the manuscript.

But, what do these TEI fragments have in common? And, how are they related with the original document? It is evident they both share the same plain text: removing all tags for a while, we would have a raw content we could define the *no-markup content* (actually, no-markup document cannot exist as discussed by Coombs et al.[CRD87]). Complex structures can be built over such a minimal level, according to need and preferences of the author, the encoder or, even, the reader (when they are added later on, after a further mark-up process). Those structures are meant to make explicit the meaning and the logical organization of content, that otherwise would be difficult to be interpreted and processed.

Traditionally two components are clearly identified in a document: the *content* and the *structure*. They are distinct but interconnected, since they have different goals and roots but they cannot live without each other. All the more so, the structure is a vehicle used to get further operations ready, as the abovementioned analises, as the transformations in different formats and so on. Note that they both have the same relevance, but they are two different and complementary dimensions: content expresses the units of information, while the structure expresses the relationship among them. An interesting example, borrowed by DeRose et al.[DDMR87], explains better such difference: the authors showed two fragments of the same text (by Brian Kernigan) translated in english and french, and argued that the structure of the document was likely to remain intact, while words and syntactic elements (the content) changed. In the same paper, the authors highlighted the importance of the structure as a means to make explicit the hierarchical order of elements and their relations (see section 1.3 for a deeper discussion about the OCHO model they proposed): containers and subcontainers help authors in understanding content and make simpler and more powerful future applications.

A different perspective can be alike useful to understand the roles of content and structure: the structure as a *contact point* between the *producer* and the *consumer* of content. I used the generic terms 'producer' and 'consumer' to extend the scope of my definition: in the case of the manuscript, Newton was the producer while his contemporaries (and future readers) were the

consumers; all the scientists who copied the document were a producers of a new document, that used the same structures of the original one; in the same way, the TEI encoder who created the first fragment translated the original macro-structures into an electronic format; on the other hand, the second encoder used different structures, addressed to different consumers. A different structure is a different angle over the same content: on the basis of the actual information a producer wants to transmit, its granularity and complexity, as well as its further developments and processes, he/she chooses some structures instead of others.

Skipping presentational aspects for a while, the distinction between content and structure does not settle the matters. Each document, in fact, is actually written in a specific languague; in particular, electronic documents are encoded by following different syntaxes, tag names and grammatical rules. On the other hand, structures are independent from specific languages and formats: they are generic constructs used to make explicit the overall organization of a text. Consider, for instance, the following HTML fragment, encoding the example:

```
<html>
...
<h1>Of Colours</h1>

<p>
1. The rays reflected from Leafe Gold are yellow but those transmitted are blew, as
appeares by holding a leafe of Gold twixt your eye &amp; a Candle.</p>

<p>
2. Lignum Nephriticum sliced &amp; about a handfull infused in 3 or 4 pints of faire water
for a Night the liquor (looked on in a cleare violl) reflects blew rays &amp; transmits
yellow ones. And if the liquor being too much impregnated appeares (when looked through)
of a darke red it may bee diluted with faire water till it appeare of a Golden Colour.</p>

<p>
3 The flat peices of some kinds of Glase will exhibitthe same Phaenomenawith Lignum
Nephriticum. And these Phaenomena of Gold &amp; Lignum Nephriticum are represented by the
Prisme in the 37<super>th</super> experiment as also in the 22<super>th</super>  &amp;
24<super>th</super> Experiment.</p>
...
</html>
```

The structure is the same of the original manuscript and the first TEI text. The overall organization in subcomponents (paragraphs) remains intact: what changes is the syntax used to express that structure, but the abstract elements do not change. Such simple example shows how documents can be written in different syntaxes and share the same internal organization of content. Similarly, an HTML page with a different layout, a MS Word file or a PDF file for the same work (divided in paragraphs) has the same content and structure of all the previous examples.

Those documents are evidently different for their formatting and layout too. A third irremissible layer of a document is composed by all those information added to make the document appealing, interesting and consistent with typographical needs and preferences: the *presentation*. Colors, spatial organizations, fonts, images and whatever embellishment complete the logical structure of the document with presentational aspects.

The relation between structure and presentation (and their separation) deserves a deeper analysis, that goes beyond the classical vision of 'producing documents of pure content, without any presentational aspect'. It strongly depends on who or what accesses a document: human beings, in fact, cannot read documents without presentation while automatic processes or agents can. We humans need something more to perceive and read a document, something physical and sensory, something that shows inherent and internal structures. So, the presentation is an extra layer, strongly interrelated with the structure, that adds and extra meaning and help us in perceive and comprehend text. Kasdorf[Kas98] summarized such a vision at best: "print pages have been communicating structure for centuries - visually". Coming back the producer/consumer metaphor, I would say that the presentation is the contact point between human consumers and the structure of a document, as much as the structure is the first contact point between content and consumers.

I found a very interesting example in a Hillesund's article[Hil02]. Actually the author used it in defense of an opposite theory, that separating content structure and format is an illusion, since they both are necessary to fully express the meaning of a text. On the contrary, I agree they

are closely interrelated but I see them as built on top of each other, since presentation strengthens what is already expressed by the structure. Hillesund argued that meanings in tables are expressed by the combination of data (words and figures) and visual layout. The cells can at the same time show data values of two different variables: this information is ”shown” in a visual and two-dimensional way, so that users can easily compare values within different cells. The strength of the table, according to the author, is just such a sensory perception of users and the subsequent simplified access to those data. But, is that information really ’visual’? Can it be really considered presentation? What makes tables strong in displaying those data is just their structure, their internal organization in cells and items straight accessible and comparable. The fact that we visually read a table very fast is an extra value, dependable on our perception. Moreover a table with specific colors, fonts or paddings makes it easier (or more difficult) the reading but it structurally remains a container of data. Let us think about people with visual disabilities, that access an HTML table by using screen readers: they do not use visual hints, but they exploit the logical organization of the table (that is possible only when authors have correctly organized and marked it and, unfortunately, that is not so frequent).

Sometimes presentational information seem to be indistinguishable from structure. Even in that case, what we really need is the deep and logical meaning hidden behind that presentation, rather than the presentation itself. The formatting remains something more, extremely useful to make explicit the structure but still unnecessary. Consider a different example (from Hillesund's article[Hil02] as well) of a norwegian fragment, enriched with some visual information.

Åseile inn i fremtiden

Livet i en seilbåt eller robåt gir folk anledning til å gjenerobre den sakte tiden, den som i våre dager er i ferd med åbli en mangelvare.

Forbundet KYSTEN har formulert som vesentlige målsetninger ågi vern til kystkulturen, ta vare pådet som var i ferd med ågåtapt, i tillegg til åstyrke vår identitet som kystfolk.

Denne fortidsorienteringen har sine kritikere. Både blant ekstrem-urbanistene som Erling Fossen og blant samfunnsforskere har man sett tradisjonsorienteringen som nostalgiske klynk etter en svunnen tid.

People who do not speak norwegian cannot translate the text, but they can guess its overall organization, at least. It is not difficult to find the title, to see that the first paragraph is probably an introduction or a blockquote, that other paragraphs are normal, with a fragment in upper-case (probably a name). Yet, all those information can be captured thanks to presentational clues provided by the author, but those clues does not substitute and cancel the structural value of the text. The same text can be transformed, for instance, into an HTML with a completely different layout: the logical organization does not change, although graphical hints are completely different. A mapping between structure and presentation always exists (no document can be displayed without a presentation), but they are two dimensions built on the top of each other: presentation highlights structure, which is anyway inherent in the content organization made by the author.

Actually, a *presentation* can be further split in different subcomponents, which describe separately the most relevant elements of layout and formatting. A very common classification identifies two main dimensions: *space organization* and *skins*. The space organization describes the positions of each structural object in the final document, the distances among boxes and shapes, the dimensions of each object and so on; a skin describes properties like colors, fonts, backgrounds, textures and so on. A deep discussion about further segmentations of a layout is out of the scope of this work, whose paramount goal is investigating the relation between content, structure and presentation. However, a clarification is needed: space organization and skins are independent from a specific language but they express general concepts. As the structural elements can be translated into HTML tags, TEI constructs or paragraphs in a manuscript, so presentational information can be translated into SVG constructs, XLS-FO primitives, formatted blocks on a piece of paper and so on. As it happens for structures, a third dimension is needed to describe the pre-

sentation of a document: the *language* actually used to instantiate abstract formatting elements like boxes, spaces, lines, images and so on.

To summarize, I have segmented the manuscript into three clearly distinguished dimensions: *content*, *structure* and *presentation (space disposition + skin)*; sideways I placed the *languages* actually used to express all these dimensions. The same segmentation model keeps being valid in other contexts too.

1.2 Heterogeneous scenarios, a common denominator

One of the main objections that can be raised against the classification proposed in the previous section says that the segmentation applies only on some specific (and simple) documents. As discussed before, some researchers argued that a clear distinction among those components is an illusion, since components are interwoven and indivisible (as claimed by Hillesund[Hil02]). I believe it is always possible, even in contexts where graphic elements are heavily intermixed with content, and the presentation is the most evident part of a document.

The key-aspect is the nature of the segmentation process: what I want to do is identifying elements of each segment, understanding their inherent role and making explicit their relation and interaction. Segmentation does not aim at dividing a document into sub-modules that can be combined together in order to obtain a document **identical** to the original one. Rather, it aims at dividing a document into sub-components that can legitimately express the same meaning, the same organization or the same overall graphical impact. Some loss of information is inevitable but it has to be limited and acceptable, taking into account the context where segmentation is applied.

Let us discuss how different documents can be segmented according to the ideas discussed so far. Consider, first, the film poster shown in figure 2:

Figure 2. Segmenting a film poster

The first goal of this kind of document is capturing the attention of the audience. For this reason, the emphasis is put on breath-taking sentences, colors, images, space organization and so on. However, the poster carries information about the film, as title, main actors, production and so on. A set of structured information were put together and, then, organized into an appealing and good-looking poster. Probably the creation process itself followed the same schema: film

data were communicated to a professional graphic designer, who organized them according to suggestions, requests and preferences. Neglecting information about the production (only for space limits), users could write a XML fragment describing the film:

```
<FILM>
<TITLE>Kiss Kiss, Bang Bang</TITLE>
<SUBTITLE>Sex. Murder. Mistery. Welcome to the party.</SUBTITLE>
<ACTORS>
 <ACTOR>Robert Downey Jr.</ACTOR>
 <ACTOR>Val Kilmer</ACTOR>
</ACTORS>
</FILM>
```

Although the differences between these documents are evident, their intimate *structure* does not change: obviously the second one cannot be used to present a film, the first one is not adapt for data extraction and they are actually two well-distinguished resources. However, the same information are reported by both of them: they are united not only by the text they contain, but also by the way this text is grouped and organized in containers, lists and sub containers. Then, they share content and structure, while they have a manifest different presentation.

Similar considerations can be extended to a different kind of documents: the magazines. Consider the cover of an art magazine shown in figure 3. Images, colors, fonts and positions of the objects were designed to grab the readers' attention. On the other hand, such a cover hides references to the most important articles of this issue, besides information about the magazine (price, issue number) and some advertising on the top. In other words some structured content are highlighted by using typographical effects and clues. The central area of the document can be translated into HTML as a sequence of `P` and `SPAN` carrying basic information. Presentations are very different but the segmentation process does not change. Authors can build different documents that show the same set of basic information, by interchanging presentations over the same content and logical structure.

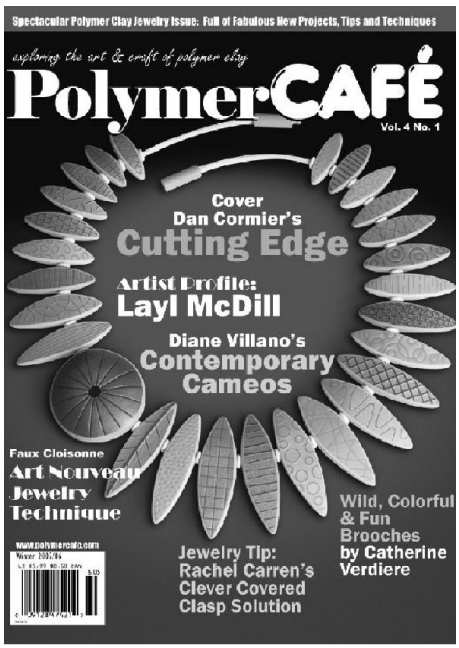

Figure 3. Segmenting a magazine cover

The role of the structure is describing the logical organization of a document and the relation among objects. Consider a page of a newspaper, in figure 4. The page is composed by three main areas: the central one with the main article, the left one with some announcements and

the third on the right-bottom side with an advertisement. Such division in not merely a visual distinction but it hides a deeper segmentation: the page is *structurally* composed by a title and three containers; each containers can be divided in sub containers (the central one with a subtitle) or contiguous paragraphs; moreover some inline elements are highlighted. The number of columns, the color of backgrounds, the dimensions of characters are information that do not add any logical meanings. Rather, they are visual hints which help human readers to perceive and read information.


PALOS VERDES PENINSULA NEWS

SATURDAY, MARCH 18, 2006

Arts & Entertainment

Local Irish Band Celebrates St. Patrick's Day with Military Men and Women Overseas

thebridge

Figure 4. Segmenting a newspaper page

1.3 Content, structure and presentation: are they enough?

From the previous analysis the only constituents of a document seem to be content, structure and presentation. However, such a three-layer distinction is not new in the literature. Glushko and Mcgrath[GM02] presented an essentially similar 3 level analysis, with slightly different terminology and definitions. Three level analyses like this are generally well accepted in the SGML/XML literature on documents:

- *Content components*: the pieces of information in the document; the "what is it" information, or the "gray matter"
- *Structure components*: the arrangement of the content, the "where it is" information, or the "skeletal matter".
- *Presentation components*: the formatting or rendering of both structure and content components; the "what does it look like" information; much of the time it "doesnt matter" except as it helps to identify components of the other two types.

It is not difficult to find examples where such a classification is not enough, in particular considering the increasing importance of interaction and dynamic behavior in some contexts, like the World Wide Web. Most HTML pages go beyond the static model of documents, and contain features that need additional model elements. Such dynamic characteristics have often been treated as a sub-aspect of presentation, but clearly involve a variety of special issues. Consider for instance the portal home page shown in fig. 5. The structured content of that page consists of all paragraphs of information, links to surf the portal, meaningful images like those associated to each main event (in the middle of the page), while the presentation is the spatial organization of content, the use of some colors and logos, and so on. But, what about the search engine inferace in the top area of the page? It clearly indicates the presence of some non-static content and the need of modeling reactions and events associated to users input.

Figure 5. Segmenting a web portal home-page

Many web pages contain programs with sophisticated logic and variable display functions, frequently linked to the server back end by Ajax or other techniques. These examples violate any simple notion of a static marked-up document, since both content and presentation could depend on arbitrary computation and user interaction. A new dimension is then necessary: the *behavior*.

However, dynamic content is not only a prerogative of the World Wide Web. For instance, Lumley et al.[LGR05] proposed DDF (Document Description Framework), a solution that adds programmatical behaviour to the documents in order to obtain flexible and variable data printing. A DDF document is composed of three different sections: data, logical structure and presentation. Structure and presentation are defined using templates, that transform respectively the data into a structured XML format, and the structured document into a suitable presentation format. Whenever a document is evaluated, the templates are applied, "moving" content from one section to another one. In the end, when the presentation part contains the transformed data, the document can be exported into a final version.

Active documents represent another example of applying behavior to the static content. Bompani et al.[BCV99] proposed a flexible mechanisms to activate different behaviours on the same document, the *displets*. A displet is a special rendering (even if its applicability goes beyond rendering) module that basically performs some operation over the plain content of a document: by activating one displet instead of another an active document may then be displayed, be printed, perform computations, animations, and so on. Rather than being only a static resources, such documents carry information about how to handle their content.

One more dimension is required to model (digital) documents: *metadata*. Metadata are all those information about a document that allow authors, managers and readers to make sense of its content in relation to other documents of the same kind, other documents related to this one, other versions or variants, other external resources. Uses of metadata are very different: they first help users to identify resources in wider contexts, to organize them in (sub)classes, to easily search them (by using simple techniques like keywords, or complex data mining algorithms), to archive and preserve them over a long period. However metadata exist very long before digital documents, as witnessed by the catalogs and archives found in ancient libraries; with the advent of digital documents they are empowered, applied to several contexts and studied as independent subject.

One of the most widely deployed metadata standards on Internet is Dublin Core[NIS01], a set of elements for cross-domain description of on-line resources. It is widely used to describe digital materials such as video, sound, image, text, and composite media like web pages. Comprising

15 categories of metadata in its core version, this standard can be extended and customized for specific domains (an example is the Qualified Dublin Core, which adds three levels to the current metadata set) and has been adopted in very different projects about art, archeology, medicine, chemistry and so on. Digital and print libraries have more complex and specialized needs for bibliographic description, and often use standards such as Marc21[MAR99] for bibliographic data, or Premis[pre04] for archives, registers, indexes and any resource that supports bibliographic search, or IFLA-FRBR[IFL97] which describes, among other data, events and actors involved in managing collections of bibliographic material. Actually any application domain suggests a set of specialized metadata, such as ID3[Nil98] for music, and many others.

An interesting classification is about the position of metadata with respect to the document. Metadata can be inserted as external resources, completely separated from the original one, as it happens for DBMSs or XML/RDF/OWL information; on the other hand, they can be located in specific sections of the document itself, as in the case of the HTML META tag or the TEI header TEIHEADER. A different approach is proposed by microformats[CTK+05] which directly embed metadata within text by merging them with the actual content, and by expressing metainformation in proper in-line tags and attributes.

This dissertation focuses on content description and analysis, rather than the important issues around metadata standards and representations. These issues will surface in the discussion of implemented systems, but are orthogonal to the work discussed here.

2 A document segmentation model: Pentaformat

All the abovementioned examples drive us into the first result of my thesis: a model to segment documents into reusable assets. The innovation does not rely on the segmentation itself, rather on its simplicity and wide applicability to a broad range of documents.

Documents are traditionally segmented into content and presentation and, although some opposite opinions exist as discussed in the previous section, researchers and professionals agree on advantages of such approach. I propose to refine that distinction by identifying five components that can be extracted from any document, regardless of its actual layout and presentation. The model is called Pentaformat:

- *Content*: the plain information made of text and images (I mainly focus on these elements, and leave out audio and video only for the moment).
- *Structure*: the labels used to make explicit the meaning and the logical organization of the content. Structure is meant to indicate the role of text elements and their relations, and to make a text interpretable and processable. Both structure and content constitute the basic information written and organized by the author.
- *Presentation*: the set of visual and typographical features added to maximize the impact of the document on human readers. Presentation is built over the structures and aims at strengthening what is inherently expressed by structured content. It is not a "useless" layer, rather 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 increasing importance of interactivity and dynamic content within digital documents.
- *Metadata*: the set of information about the document. They allow authors, managers and readers to make sense of documents in relation to other documents, other versions, or other external resources. They are then meant to make resources searchable, indexable and manageable within wider contexts.

Figure 6 shows my segmentation model, emphasizing the role of each abstract consituent. My claim is not only that *any* document can be considered as the integration of those five dimensions,

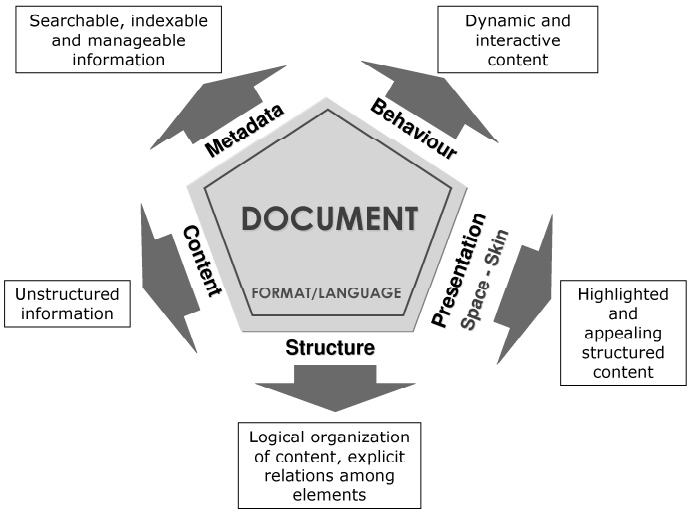

Figure 6. The Pentaformat Model

but that they are clearly distinguishable from each other, and can be interchanged and reformulated. In order to better explain the nature and impact of Pentaformat, some properties of these dimensions are discussed below:

- *Logical separation*: each dimension is a partial perspective on the same document. Each dimension expresses specific information, derives from specific competences and has a specific role for the overall meaning of the document itself. Note that talking about logical separation does not mean these components are always created separately and by different users (on the contrary it is very common to find them intermixed); rather, it means they can be *abstracted* and separated *a posteriori* to express different kinds of information about the same source document.

- *Mutual connection*: these dimensions are also strongly connected. They are built on the top of each other, and they "work together" for the overall meaning of the document. For instance, structure organizes the plain content, presentation adds typographical information to the structure, as well as behavior and metadata deals with them. When examined in isolation, none of these dimensions gives a complete picture, as they each omit critical information about a document, regardless of the ongoing disputes as to what is essential for a document to be described.

- *Context-based relevance*: no hierarchy is imposed *a priori* over these dimensions, but they are equally important from a theoretical point of view (except for the content itself, which is essential to any use of the document). It is the context that determines their relevance and replaceability: for instance, presentation is very important for professional publishing but can be discarded in data mining applications, as well as metadata can be neglected in pagination processes, or dynamic information are useful only in some contexts.

- *Context-based interchangeability*: depending on the context where a document is actually used, these components can be substituted. Examples are fitting structured content into a completely different presentation, or expressing a set of metadata expressed in a different vocabulary, or transferring a set of dynamic behavior onto a different platform and so on. The point is that the obtained document does not carry the same information of the original

one, but a new information relevant for that context. When a user needs to update the content of a page, for instance, presentational aspects can be changed; equally, when a graphic designer wants to change a layout the actual content is not relevant, and so on.

- *Language independence*: every one of the five dimensions can be expressed by different languages. The issue of languages and their semantics interacts with every dimension of the document model in an actual system. From a theoretical point of view, however, the actual instantiation into a specific format does not influence the meaning of that information. The capabilities of a specific language limit what can be encoded with that language. For instance, structural elements (such as paragraphs, lists, tables, etc.) can be translated into HTML tags, TEI constructs or any other encoding language, as well as presentational information (box, lines, colors) can be translated into SVG constructs, XLS-FO primitives, formatted blocks and so on. The same considerations can be obviously extended to behavior and metadata. In conclusion, a cross dimension is necessary to complete my model: the *language* each dimension is expressed.

At this point, my claim should be more clear. I want to separate and extract all constituents of a document so as to be able to reformulate part of them, or reuse some of them for different contexts.

3 The need of segmentation

Some benefits of the Pentaformat model are evident, and shared with similar proposals, other are more specific and need a deeper investigation:

Complexity reduction: segmenting a document also means segmenting tasks and objectives in managing it. Users do not need to deal with many intermixed components but they can concentrate on a single aspect. The scope of analysis, implementation and test is then reduced in sub-problems, even if they can be very difficult.

Role distinction: a direct consequence is the possibility of designing workflows and systems based on a strong distinction of roles. Each actor involved in producing high-quality output is usually expert on some aspects but completely unaware of others. Consider for instance the publishing of a professional book: authors write content, editors revise it, typesetters paginate it, and in case web designers transform it into appealing site pages. Making all these actors work together is a richness and motive of success. Each of them can specialize on specific aspects, so that costs (and time) of production and training are reduced, as well as the quality of results is improved.

Reuse: segmented components can be re-used in new documents or moved from one document to the other. Many different scenarios can be envisioned: moving content from paper-based documents into on-line versions, re-flowing content in small devices like PDAs, uniforming presentation of heterogeneous web sites, removing behavior information in static contexts, enriching metadata constituents so that the document can be correctly placed within workflow processes; and such list can go on and on. The point is that users create subcomponents once, which can be integrated in different applications with little effort.

Composition: heterogeneous as well as single sources can be combined to obtain new documents. A meaningful example can be found in professional printing, considering how miscellaneous books (whose chapters come from different books and editions) are currently produced. Each chapter originally has different formatting properties, different page numbering, different organization of footnotes. Editors collect those chapters, basically remove presentational aspect, revise content and pass them to typsetters, who actually paginate the final book. Such process is implicitly based on a segmentation and normalization process. Some researchers argued that reflowing content 'as is' is almost impossible since the content is completely embedded in the whole context and thread of a book (see for instance the position of Hillesund[Hil02]). I agree with that position but here the perspective is different, since it focuses on the technical feasibility of segmentation rather than on editorial interventions required to the authors. Undeniably

when authors write content with dependencies between chapters, with constrained references, or with a predefined order, miscellaneous sources are difficult to be combined; on the contrary, when content is modular and designed taking these ideas in mind, composition becomes an extra value.

Automation: besides extending the scope of publishing , a segmentation model improves the production process itself. Consider, for instance, the most common approach to generate e-learning content: first, the author produces initial material in a source format (usually created with personal productivity tools) and then this collection of unrefined materials is processed with *ad-hoc* tools by a staff of experts. These experts transform material into web pages, organize learning objects and add metadata. Content updates (as small-time typo corrections) need to be performed directly on the final learning object, by exclusively using the authoring tools included in the platforms. Even little modifications require many steps to be performed and intermediate documents to be produced. On the contrary, a segmentation model and an automatic composition tool allow users to save costs and time, since final documents can be directly produced from sources files and modifications can be performed on them.

Adaptability and portability: segmented documents are independent from specific platforms. Customizing and re-arranging subcomponents, they can be easily ported in different formats, in order to maximize their diffusion and impact. For instance, web pages can be displayed in small devices by removing sophisticated presentation, metadata extracted from web pages can be exploited by search engines, conversion rules can be derived from templates and applied on different pages, embedded metadata can support data mining, and so on.

High-quality output: the final effect is an improvement of the overall quality of the documents. Uniform, well-studied and interchangeable look&feel, for instance, can be ensured over documents by working on single subcomponents and recombining them. Obviously such quality is not a free-of-charge consequence of document segmentation, but it requires a lot of implementation effort. What is important is that a rigorous and flexible model makes that implementation possible and allows designers to hide complexity for final users, without sacrificing the quality of the results.

3.1 What matters for authors: structured content

The Pentaformat model is based on the notions that every segment has its own elevance and no hierarchy can be imposed *a priori*. This dissertation will focus on the dimensions of *content* and *structure* and, indirectly, *presentation*. These dimensions have been chosen as the primary ones where an author works on a document. The goal of this investigation of when and where content and structure can be managed in isolation is to empowers authors with a well-structured process that produces well-structured content.

The example of the wiki editing paradigm[CL01] is particularly meaningful. Figure 7 shows a page from WikiPedia[SW01] during an editing session. The fact that authors can change only the raw content of a page is one of the strenghts of wikis, since it allows unexpert users to easily add content, to revise versions, and to work only on the actual information of a page. Apparent limitations on presentation, basic text objects, etc. are actually the reason of the wikis success. Most authors are in fact more interested in editing and collaborating on simple content, rather than mastering complex tools and technologies to manage other features. It is no accident that the content/format separation so deeply embedded in the community (as discussed in section 1.2) reflects a separation of competences and spheres of interest between *content author* and other roles.

In many cases authors do not know how and where their content will be actually delivered. Professional content management systems (CMSs) are examples of such approach: for instance, reporters write articles (and send them in any format), which are later paginated and distributed. Similarly, most of the e-learning authors write content without knowing details of the final publication on a LCMS, as well as PMS (Portal management systems) present raw information from different sources into new layouts. On the same line, smaller CMSs apply templates to plain content written and pasted by the authors through specific interfaces.

Figure 7. Editing a wiki page

The interest of authors for content purged from presentation can be observed in another context too: *web customization*. The term 'customization' indicates here the possibility for users to personalize web content, regardless of access permissions, and store personal variants in external databases. Whenever the same user accesses the same page, a customization systems substitutes the original page with the customized version. Users can then annotate, and compare different resources in a more powerful and democratic environment. Chapter 6 will discuss the importance and feasibility of web customization in detail; what is interesting here is that content customization alone is profitable for users.

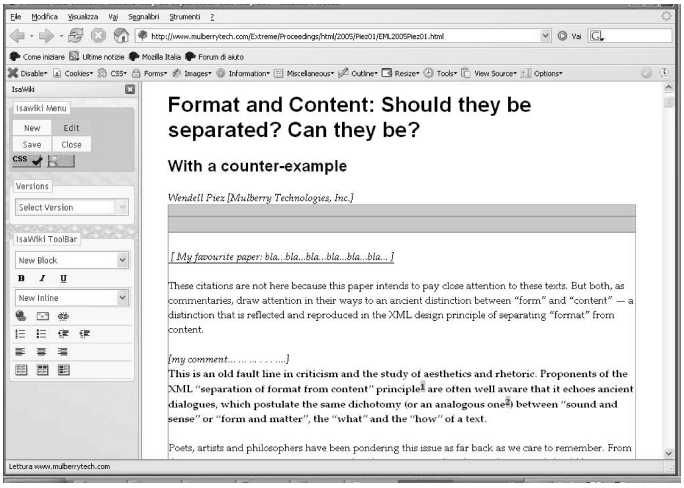

Figure 8. Commenting a segmented paper

Consider, for instance, an user interested in commenting on a paper written by someone else and published as HTML. Such a customization help him/her to review the paper, to develop his own ideas, to emphasize connections with different works, to share opinions with other users and so on. Whatever layout will be finally applied to the paper, whether or not it will be converted in PDF, whatever format it was originally written, user need to actually deal with the only original

structured content. Figure 8 shows an interface of the IsaWiki systems (discussed in chapter 6) to customize such a paper, where content areas are bordered and editing facilities are activated only on those areas. The fact that users cannot change the layout of the page, or the title and metadata does not limit the power and usefulness of customization.

A different application of content customization is as means to review, select and comment on a list of products (for instance owned by a rival firm) published on a public web site. The most relevant information are the units on sale, their prices and their descriptions. The layout, the order of elements, the navigation schemas are out of the scope of customization. In figure 9 relevant areas are again bordered and highlighted in order to let users to customize them, without touching logos and other presentational features.

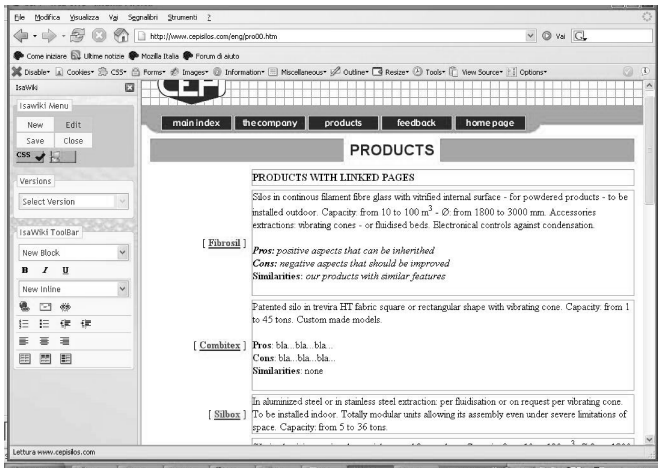

Figure 9. Customizing a segmented web-site

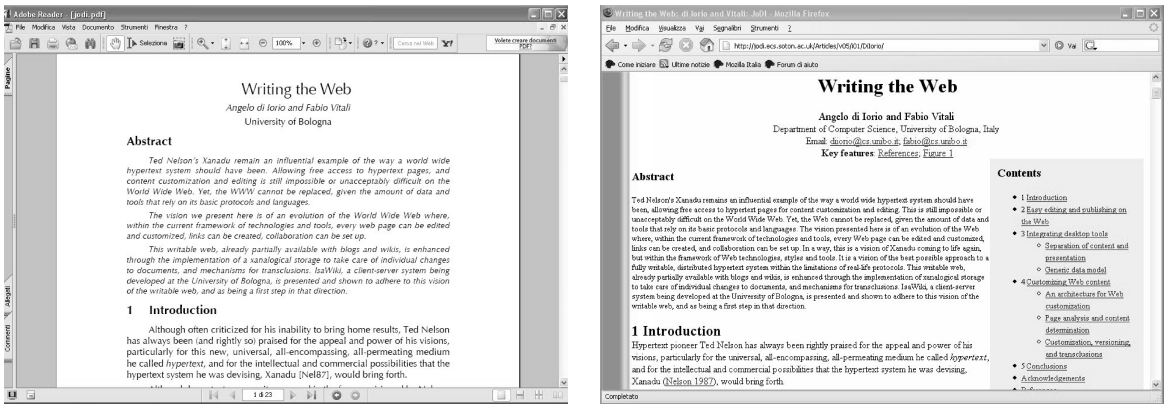

Figure 10. Content-based diff-ing on a scientific paper

Also the differences between documents can be calculated on the only structured content, after discarding presentational aspects. That solution is both powerful and indispensable when documents are stored in different formats and layouts. Figure 10 shows two versions of a paper I wrote about IsaWiki[DIV04]: the left side shows the submitted version in a raw formatting, while the right one shows the final version formatted according to the journal specifications. Which differences might we be interested in, for instance after some years? The content we added or removed would probably be the only relevant information in that context. Then, performing a

diff between these documents can be limited to performing it on the only content extracted from both of them. In conclusion, many contexts exist where structure and content actually play a leading role and limiting the authoring process on those dimensions does not limit the power of the writers.

Chapter 4

Pattern-based Segmentation of Structured Content

According to the Pentaformat model discussed in the previous chapter a document is the result of the interleaving of some (five) distinguished but interconnected constituents. The Pentaformat does not impose any hierarchy among those segments but simply states their distinction and mutual relationships. It's the user (and the context) who suggests a specific hierarchy among components. From authors perspective, however, the top of the hierarchy is held by the content and structure dimensions and most operations are performed on that information.

This thesis focuses on these two segmented dimensions. In particular, the objective of this chapter is discussing a pattern-based approach to express and normalize the structured content of *any* document. From the multitude of languages, formats and documents we daily work on, we might conclude that a huge amount of complex and diversified structures are needed. Although that complexity seems to be unavoidable to express a so huge variety of information, a (very small) subset of structures/patterns is alike enough to express what most users need. My point is that it is always possible to write simplified documents (or to normalize existing ones into simplified versions), that use only a limited set of structural objects and composition rules, but keep on expressing the same basic information. A first discussion about such pattern-based approach can be found in [DIGV05]. My approach is strictly related with the Pentaformat. Patterns, in fact, are meant to capture the structural elements of a document, and to express two of the five dimensions of the Pentaformat segmentation model: content and structure. Segmentation is by definition something that happens after the creation of a document, something *descriptive*. Then, a deeper analysis is first of all needed about the distinction between descriptive and prescriptive markup languages.

1 A descriptive perspective: too many structures?

Quin[Qui96] outlined some important features of prescriptive and descriptive approaches in designing markup languages: a prescriptive DTD may be designed to create new material or to mark up existing material, and prescribes a set of rules which all matching documents must follow; a descriptive DTD is used to create an electronic version of material that already exists (of course, a descriptive model may also be used to create new documents) and describes structures that exist, rather than to force any particular structure. In a prescriptive context, if a document contains a structure that cannot be described by a DTD the document must be changed to fit the DTD. If something should occur in a document that a descriptive DTD does not permit, it is the DTD that must be modified.

The choice between these models primarily depends on the relation between the process of actual writing a document and the process of adding markup information. In a sense prescriptive models give the most expressive power to the document designers, and make document authors

subject to the power of the constraints, while descriptive models reflect the fact that sometimes document authors work and have worked independently of the desires of the document designers, and thus the latter have to accommodate variations, exceptions, differences, etc.

By adopting a descriptive model rather than a prescriptive one, the role itself of validation changes. Piez[Pie01] distinguished two kinds of validation: *strict* and *loose*. The traditional way of conceiving validation is "strict", because validation is used as a "go/non-go" gauge to verify *in advance* whether or not a data set conforms to a set of requirements. The example provided by Piez explains very well the role of such a validation: the publishing process can be likened to an assembly line and validation is a control phase that prevents errors and makes the whole system work. When a document fails validation, there is something wrong with it, something that has to be changed in the document itself. Then, validation is a *pass-or-fail* exam, whose output is the capability to go forward in the publishing chain. From that perspective, strict validation is useful (and sometimes necessary) as a means to split a complex job into sub-activities, that can be accomplished by different actors with different skills and facilities.

Even if less frequent, an opposite perspective is alike interesting: using validation to describe document and to capture *a posteriori* structural information about a text. It might be important to trace those features of the text important to the encoder, rather than those constraints essential for subsequent operations over that text. Piez defined such a process as a *loose* validation. Loose validation aims at capturing what a text is, instead of what a text should be. Then, it is meant to be an analytic instrument: while a strict validation is a "valid/invalid" checker, loose validation is likened (by the same author) to a caliper that measures some properties and qualities of a document.

In such a context, the relation between documents' instances and schemas changes. A schema is not something that exists before an instance, as a set of rules to be followed; rather, it derives from instances, as a *ex post facto* expression of what can be discovered from them. As a consequence, a schema for loose validation is not composed by fine-grained declarations that capture variations and exceptions, but it is composed by generic rules that capture the overall meaning of a set of documents. Constraints are relaxed, while repeated structures and patterns are more evident.

Both strict and loose validation are useful. What is important is designing languages and schemas by keeping in mind their features and differences, and applying them in right contexts. A fully descriptive schema cannot be used as a means to verify minor imperfections or to impose structures to new documents; on the other hand, a schema for strict validation is not suitable to express common features of documents, discovered *after* having analyzed actual instances and data. To create a segmentation model, loose validation is then more interesting.

My approach can be further explained by discussing how to model anomalous data. Birnbaum and Mundie[BM99] presented a very interesting case: in the context of encoding an historical edition of an english dictionary in SGML, they asked themselves how an entry that violates the structural conventions of that dictionary could be marked up. Three main approaches were discussed: editorial correction, escape hatches and a loose DTD. A first solution would be intervening manually to change the text to fit the normal model; this is clearly unsatisfactory since the original entry is not actually recorded, but becomes something completely different. A second solution would be designing parallel structures able to model differences (for instances, in TEI the element `entry` is used to model well-structured lexical entries, while the element `entryfree` does not impose rules and can be used to markup up bad-formed entries). The authors, argued that such a solution obscures the fact that an entry is actually erroneous: it considers that anomaly as an appropriately unusual or unconstrained element, while the designer of that dictionary schema would not have actually consider that entry as valid. A third approach would be using a loose DTD that relaxes some constraints and makes also the bad-formed entries valid. The author did not accept that solution because it does not capture the structural validity of most of the entries, and lose a relevant piece of information.

I am not interested here in the solution proposed by Birnbaum (he proposed to maintain the text in a valid SGML document from which the invalid one could be automatically derived at any time) but I find his example useful to highlight differences between those descriptive approaches.

Apart from the first one which is clearly unacceptable, two opposite goals underpin the others: an escape hatch wants to keep *all details* of each entry in order to reproduce a faithful copy of the original resource; a loose DTD wants to describe *all entries* without focusing on minimal differences and variations among instances. A loose DTD allows designers to gather all entries in a common area described by the DTD itself. My descriptive approach roots in that collective and generic idea of validation.

On the basis of these remarks, how should a descriptive language be designed? The temptation to over-design (i.e., to impose too many constraints on document structure, as if we were in a prescriptive situation) is strong, and may lead to situations where actual documents cannot fit the structure because they are too different from the "natural" candidates. On the other hand, the temptation to under-design (i.e. To give up and say "anything goes") is also to be fought, because this would lead to major differences in markup of the same documents given by the lack of absolute standards to refer to.

In practice, validation languages are too powerful and easily lead to overdesign. Murata et al.[MLMK05] proposed a formal framework to study the expressiveness of the most common schema languages, DTDs, XML Schema[TBMM01] and RelaxNG[Mur00]. In particular, they used regular tree grammar theory to measure and compare validation capabilities of those languages. They first defined some regular tree grammars (and studied their associated validation algorithms) and showed which grammar captures each language in analysis. Apart from DTDs, expected to be less expressive than others, the surprising conclusion was that XML Schema is less expressive than RelaxNG. In fact, the two languages were associated to two different grammars, respectively called "single-type" (XML-Schema) and "regular"(RelaxNG), which proved to have different expressiveness.

The point here is quite different: rather than being interested in the *inherent* expressiveness of schema languages, I want to study them when used for descriptive situations. The conclusion is that they are not *intrinsically* too complex or too powerful, but they suffer such a complexity when used for descriptive purposes. The following examples will discuss how, in a descriptive scenario, structures can be shrunk into a very small subset without sacrificing the expressiveness of a document.

1.0.1 Alternatives

Consider a possible either/or situation: for instance, in an address, a document designer might decide that an address either has a P.O. Box or a street address. In a DTD like syntax, this could be rendered in a rule such as:

```
<!ELEMENT address (name, (pobox | street), city, ZIP, state) >
```

In a prescriptive document factory, this rule effectively inhibits incorrect structures to be created, and ensures homogeneity in the created documents. In a descriptive environment, on the other hand, there is no homogeneity to be sought for documents (they exist already), but rather it is important that all existing documents are marked up at best and without ambiguities.

Now two things may happen: if in the document set there is no example of a simultaneous presence of P.O. Box and street address, then this is a constraint that has no practical effect on reality, one additional check that was not needed. If, on the other hand, a document exists that has both a street address and a P.O. Box, then the rule does not allow a correct markup, and forces the document editor to find a hack around the constraints of the DTD.

A corresponding descriptive rule would therefore be:

```
<!ELEMENT address (name, pobox?, street?, city, ZIP, state) >
```

where the alternative has been transformed into a sequence of optional elements. This rule has no effect on the final markup, exposes exactly the same meanings for documents that naturally follow the stricter rule, but allows for the exception in case one exists.

Alternatives do not capture additional semantics with respect to a sequence of optional elements, but a priori exclude some situations to occur. Thus in a descriptive environment they are useless in the best cases (where all occurrences naturally follow the alternation) or a nuisance and an obstacle if an exception happens.

1.0.2 Repeatable homogeneous elements

It is sometimes tempting to insert a repeatable element within a sequence of different elements. For instance an address may include any number of telephone and fax numbers. One such rule could be:

```
<!ELEMENT address (name, ..., state, (telephone|fax)*) >
```

It is difficult to extract any meaning from the presence of several such elements directly within the address element. Certainly they have not the same role and importance of name, street, zip or state elements. Should they be taken individually or cumulatively? Does the order of appearance have an importance?

Whether the information is inherently a group, using intermediate elements is better than relying on repetition. For instance, the previous form of rule should be substituted with a more explicit and clear structure:

```
<!ELEMENT address    (name, ..., state, telephones?) >
<!ELEMENT telephones (telephone|fax)+ >
```

The telephones element (in its plural form) already hints that there will be one or many individual telephone elements inside, each of which should be considered as an autonomous piece of information.

Wrappers help in creating a strong structure and separation of concerns, give more clearness and visibility to the inter-relations among elements, and simplify the readability of the DTD. See section 4.2 for a deeper discussion about their expressiveness.

1.0.3 (Un)ordered single and multiple elements

Multiple information are very often put together in more complex structures that impose cardinality on some elements but allow others to be unlimitedly repeated. Moreover those superstructure do not impose rules over the order of elements. Consider, for instance, the content-model of the `head` element in HTML. Tags for metadata, scripts and styles are grouped through the entity `%head.misc;`, which is then combined with a single `title` and an optional `base` element in any order:

```
<!ENTITY % head.misc "(script|style|meta|link|object)*">

<!ELEMENT head (%head.misc;,
((title, %head.misc;, (base, %head.misc;)?) |
(base, %head.misc;, (title, %head.misc;))))>
```

Such a declaration is complex and difficult to be read. On the other hand, a descriptive context does not require such rigidness: once again, the goal is not pre-defining respective positions among elements, but capturing the structural and basic information captured by those elements.

In a descriptive context, designers are not interested in prescribing *a priori* where elements can appear; rather, in collecting a set of related information. Using a SGML syntax they might have declared the `head` records as follows:

```
<!ENTITY % head.misc "(script|style|meta|link|object)*">

<!ELEMENT head (title? & base? & %head.misc;)>
```

RelaxNG[Mur00] allows users to write a similar declaration for XML documents, by using the `interleave` pattern. On the contrary, XML DTDs lack that operator and child elements are not allowed to occur in any order. In a descriptive mode, such XML DTD declaration deserves special attention: the order of the elements is not meant to be an imposition to make some documents invalid, but a non-meaningful order from a semantic point of view. The real descriptive goal would be to gather a set of related information without any specific order, but the XML DTDs do not make it possible.

The previous declaration raises another issue, about the co-existence of repeatable elements and single ones. Where such a difference is not so relevant for the authors, it is good to group together all those repeatable elements. The resulting schema is more verbose (and apparently unnatural) but such a distinction makes further applications simpler and faster. In fact, the two classes of objects are conceptually distinguished and they can be managed with different policies and rules:

```
<!ENTITY % head.misc "(script|style|meta|link|object)*">

<!ELEMENT head (title? & base? & info?)>
<!ELEMENT info %head.misc;>
```

1.0.4 Conditional elements

Conditional declarations are not directly possible in DTD syntax. Although they can be easily declared with RelaxNG, or explicit solutions proposed for co-constraints like SchemaPath[SCMV04] or Schematron[Jel05], some workarounds are very commonly used to solve that issue with plain XML DTDs. Consider, for instance, the following fragment of the DocBook DTD, where `bibliography` is defined as a container for heterogeneous information, among which a title, a subtitle and an abbreviated one:

```
<!ELEMENT bibliography         (bibliographyinfo?,
(title, subtitle?, titleabbrev?)?,
(%list.class; | %synop.class; | ... | %para.class;        )*,
(bibliodiv+ | (biblioentry|bibliomixed)+))>
```

The twisted declaration of titles has a specific objective: preventing subtitles to appear without a title. Such a constraint is legitimate in a prescriptive environment but it makes less sense in a descriptive one. There, we do not need to make explicit all the relations among elements (and rules on their respective presence), but to describe which information can be provided about a given bibliography. Relaxing that constraint, users would have a simpler and more manageable schema, which validates documents that carry the same basic information:

```
<!ELEMENT bibliography         (bibliographyinfo?,
title?, subtitle?, titleabbrev?,
(%list.class; | %synop.class; | ... | %para.class;)*,
(bibliodiv+ | (biblioentry|bibliomixed)+))>
```

The previous declaration is complex and quite overdesigned as well. The first problem regards the sequence of unordered and repeatable elements following the titles (paragraphs, lists,

synopsis, etc.), which are conceptually (but not explicitly) wrapped in a virtual container. Wrappers should be used to make that relation more explicit, clear and usable. On the other hand a conceptual simplification is equally viable: do users really need a so complex structure for the entries of a bibliography? In a process of normalization, what users need to express is the list of entries and, for each of them, structured sub-information. Then, an alternative and 'radical' declaration for the `bibliography` element would be equally valid:

```
<!ELEMENT bibliography (bibliographyinfo?,title?,
subtitle?, titleabbrev?, bibliocontents?, biblioentries?)>
<!ELEMENT biblioentries (biblioentry)+>
<!ELEMENT bibliocontents (%list.class; | %synop.class;
| ... | %para.class;        )*>
```

1.0.5 Mixed content models

Mixed content models are by definition used when describing semi-structured text flows that are part of larger contexts. Very common examples are paragraphs that have meaningful subparts inside. Each individual subelement of a paragraph specifies some special meaning or style on the wrapped text. In a descriptive scenario, it seems just natural to assume that all text within a sub-element of a paragraph is also part of the paragraph.

Many counter-examples to such claim seem to exist, for instance footnotes. Usually footnotes appear in paragraphs, paragraphs appear in footnotes, but footnotes should not contain footnotes in any normal document. The problem here is that a footnote element in a paragraph should not contain the whole body of the footnote but only a reference to a more complex structure. The text fragment wrapped by, say, a `footnote` element is a landmark to indicate where the reference to the note will be displayed. The footnote itself is something else, which does not belong to the paragraph (indeed it is displayed in a different location).

Subelements should not be allowed to contain as elements data that is not part of the paragraph text flow, since this could be difficult to identify without precise advance knowledge of the meaning of the subelement itself and its further subparts. Thus the only allowable forms of mixed content models should be:

```
<!ENTITY % inline  "(#PCDATA | a | b | ... | z)*">
<!ELEMENT  para    %inline; >
<!ELEMENT  a       %inline; >
<!ELEMENT  b       %inline; >
...
<!ELEMENT  z       %inline; >
```

This is meant to specify that the content model of all elements of a mixed content are mixed content themselves (or simple text in the simplest cases), and that a block element is the only mixed content element whose content model list does not include itself (i.e., there is no para inside the inline entity).

Such declaration does not impose further contraints over the in-line elements. For instance, it validates combinations like `<b><b>text</b></b>` or embedded links (`a` element). Once again, that is justified by the descriptive nature of the schema, which is not meant to prevent non-meaningul document instances but to describe structural elements.

1.0.6 Flow text

It is very common to find DTD declarations that allows text to appear in different positions of a document. Consider, for instance, the following valid HTML, where sequences of characters are included within the elements `body` and `div`:

```
<body>
        A first text fragment.
        <div>
        A subsection containing some paragraphs and
        a further subsection:
                <div>
                A subsubsection with two paragraphs
                <p>Text in a paragraph</p>
                </div>
        </div>
</body>
```

Even if not explicitly bounded by a paragraph, those sequences can be naturally considered as text blocks distinguished from the following ones; on the other hand, `div` and `body` are used as macro-containers to organize content in nested sections, rather than low-level containers for text and inlines. The same document is better represented by an alternative structure:

```
<body>
        <p>A first text fragment.</p>
        <div>
        <p>A subsection containing some paragraphs and
        a further subsection:</p>
                <div>
                        <p>A subsubsection with two paragraphs</p>
                        <p>Text in a paragraph</p>
                </div>
        </div>
</body>
```

Any text has been wrapped in a block, while containers are used only to express structures and substructure. Note that I do not argue against using the `div` element as generic a block for HTML. I am rather suggesting to only use specific objects for specific purposes, and to strictly separate structural divisions of text blocks and text blocks themselves.

In conclusion, my point is that designers of descriptive documents do not really need all the structures provided by schema languages. In particular, the structural content can be expressed by using only a few set of *patterns*, that "describe" the basic information of the document.

2 Why (XML) patterns and what for

Patterns are widely accepted solutions to handle problems which recur over and over. Their inventor Alexander defined a pattern as "a three part rule, which expresses a relation between a certain context, a problem, and a solution"[Ale79]. The basic idea is then to capitalize previous experiences, in order to re-propose solutions for similar contexts. Alexander was an architect and proposed repeatable solutions to build gardens, streets, buildings, etc. Soon researchers and professionals understood how advantages of patterns in terms of reusability, flexibility, easiness of creation and modification could be extended in other fields too.

In particular the community of software engineers and object-oriented programming experts looked at pattern languages with great interests. First, Beck and Cunningham[BC87] proposed five patterns to design and code window-based user interfaces: these patterns allow programmers to organize interface in windows, to divide each window in panes (one for each sub-task) , to classify them, and to decide which actions can be performed in each sub-window. That pattern language was minimal, and the same authors announced they were extending it to include about

150 patterns. The definitive acceptance arrived when Gamma et al.[GHJV94] provided a methodical and complete description of patterns for software development. Patterns were then accepted by the whole community as a means to ensure re-usability, maintainability and reliability, and that book became a must-to-read resource for any software engineer.

The XML community has not been indifferent to the patterns, and many solutions were proposed to use them for well-structured and meaningful XML documents. Arciniegas[Arc00] divided XML patterns in three categories: patterns for *Program Design*, patterns for *DTD design*, and patterns for *DTD Implementation*. The first class includes all those solutions that can be applied when designing applications that handle XML content. They usually are traditional patterns refined to manage XML-based information, that exploit either the generality of traditional approaches or the specificity of XML data. The second class focuses on recurring problems in the overall structures of schemas (the term DTD is here used to indicate any document used to validate) and directly deal with XML structures. Arciniegas discussed two examples: the 'Choice Reducing Container' which allows users to reducing the number of choices an author has to make at any point in the DTD, by introducing intermediate wrappers, and the 'Cross-Cutting Metadata' which allows users to identify and encapsulate common sets of metadata, in order to make clearer schemas. Patterns for *DTD Implementation* are the most widely used, and cover recurring definition of content-models. Two examples were discussed again: the 'RunningText', already proposed by Graham and Quin[GQ98], and the 'Marker Attribute'. The first one is used for general textual content that may contain markup at the phrase, word or symbol level but not at the block level; it allows to logically unify the same set of basic elements, so that it is allowed everywhere text is allowed. A 'Marker Attribute' is used when certain elements need to be marked with attribute so they can be processed in a different way by a style sheet/program; it allows to differentiate the behaviors of elements, without being invasive and changing the overall structure of the element itself.

As I said, Graham and Quin[GQ98] proposed some more patterns for XML documents. They discussed the 'Generated Text' pattern, to be applied when text fragments are not explicitly present in the document source but can be produced and rendered on-the-fly (such as a title or a list-item number), or the 'FootnoteBody' to be used when a stream of text contains a reference to an external annotation or comment that will be displayed in a different flow, and the 'Text Block' to be used for contexts that stand alone (like paragraphs, headings and lists) and to ensure a clear break before and after these content objects. The authors did not claim that their classification was exhaustive, but stressed the importance of a repository of patterns where users could easily found solutions already tested and adopted by others.

While pattern languages usually propose low-level solutions which indicate how to create well-structured content models, Downey[Dow03] discussed some XML patterns to organize the overall document and some architectural choices to make that document flexible and clear. In particular, Downey suggested to use 'marshallers', that are dynamic objects able to move recursively through programming data structures and to generate on-the-fly XML trees (accommodating changes in the object internal structure); moreover he recommended to use existing tag names more than creating new ones, to include human-readable information directly within documents and to make documents' splitting into different files a simple and reliable operation.

Besides these general proposals, many researchers focused on XML Schema, suggesting specific patterns for that language. Kawaguchi[Kaw01] provided a list of things that XML schema designers should or should not do. Basically, he proposed to radically reduce the set of available features: for instance, he suggested to avoid complex types, attribute declarations or local declarations. The point is avoiding pitfalls by only using those constructs that prevent misunderstandings. Obasanjo[Oba02] answered Kawaguchi by altering his proposal (basically he restored complex types and attribute declarations), and by adding some more guidelines. He concluded saying that XML Schema is complex because was designed to handle complex problems and, although it can be simplified by only using simplest features, that means a loss of power and flexibility. Provost[Pro02] proposed some more patterns for XML Schema types: the 'Composite', to collect children (parts of a document) and express variations on each of them, the 'Instance Specialization' , to differentiate abstract models from actual instances (when restricting a type),

and the 'Peer Specialization' , to constrain the part or referenced instances to some corresponding subtypes (after deriving a type). Another very interesting and useful resource is the 'XML Design Patterns' web site[Lai00]. Seven categories of patterns are presented for a total of 28 different solutions applicable in different contexts: 'Document Roots' are patterns useful to decide what the root element(s) should be, 'Metadata' to include metainformation in documents, 'Abstraction' for containers and collections, 'Organization' about the overall logical structure of a document, 'Flexibility' to add generalization, and so on. A deep analysis of all those patterns is out of the scope of my thesis, but that resource is definitely worth being looked at.

Rather than analyzing more patterns, it is worth remarking advantages of a pattern-based approach. I list some positive aspects below:

- *Re-use*: the most evident benefit does not need much more explanation; patterns are meant to be re-used in different contexts in order to exploit legacy competences and material, to save time and resources, and to ensure quality of the final result.
- *Reliability*: patterns proved to obtain good results in specific scenarios, since they derive from the internalization of concepts, problems and solutions. The discussion about a pattern, the analysis of previous applications and the acceptance by the community make it a reliable solution. It is not enough to call a solution 'pattern' to ensure reliability. Many anti-patterns or bad habits can be found in the design of XML documents. On the other hand, solutions well documented and shared by experts are worth being adopted.
- *Organization*: patterns help users in thinking about problems; since they internalize and organize concepts, further discussions and thoughts about problems and solutions are simplified. Users do not risk to deal with an unordered magma of information, but work on a well-structured organization of data.
- *Easy authoring*: patterns transform and simplify the authoring process; authors do not need to reinvent the wheel whenever they create new documents, but they can exploit solutions that already proved to be correct. Moreover choices are minimized, since each pattern fits a specific need. What an author has to do is simply picturing the problem he/she needs to solve, consulting the catalogue of patterns and applying it to that specific context. In the beginning it will be quite difficult but, with some experience, common problems will be solved with very little effort.
- *Easy learning*: patterns can be learned with little effort, since they identify and solve specific problems, and usually have a good documentation that describes possible applications, limitations and examples. Users do not have to handle complex problems together, but they can segment problems in sub-units. Indeed many patterns aim at separating documents into more manageable fragments, and at applying recursively pattern-based solutions.
- *Easy composition*: patterns are designed to be composed into complex structures; it does not make much sense mixing well-engineered local structures, with ambiguous and bad ones. Then, patterns designers usually provide set of solutions that can be combined together toward high-quality documents. The composition of patterns is then either simple or expressive.
- *Easy transmission*: pattern languages 'standardize' a way to describe both problems and solutions; then transmitting experience and know-how is simpler, since a pre-defined and well-known schema can be exploited.

3 Patterns for documents substructures

The core of my work is presenting a set of patterns able to express the most used and meaningful structures of digital documents. In order to explain this specific pattern-based approach, it is useful to remark which is the (strict) relation between those patterns and the Pentaformat

segmentation model. The patterns I am proposing do not aim at describing all the dimensions of a document (presentation, metadata and behavior are neglected) nor at capturing minor differences and anomalous data. On the contrary, the objective is *normalizing* the existing structures into new ones that express the same logical organization and basic content. Only two of the five dimensions discussed so far are covered. That is why a very small set of objects and composition rules is enough, though composed by the only seven patterns discussed below.

Considering the simplicity and diffusion of the patterns I propose, I describe them in a narrative style (as Alexander did with his patterns about architecture). It is not difficult to picture a methodical description based, for instance, on the dimensions used by Gamma et al.[GHJV94].

3.1 Markers

A marker is an empty element, in case enriched with attributes. Two kinds of markers can be identified: *milestones*, whose meaning is strictly connected with their position within the context, or *equipped markers*, whose meaning is expressed by their attributes (and position). A milestone is not meant to provide characterization of the text content, but to identify special rules for a given position of the text. Milestones are widely used to express destinations for fine-grained linking, as for the `A` element in HTML. They can be also used to separate (sometimes visually) what comes before a marker from what follows, as the element `BR` or `HR` do in HTML. An equipped marker is characterized by its attributes and properties, besides position. The most common example is the element `IMG` in HTML, whose actual information is carried by the attribute `@src`.

```
<!ELEMENT hr (EMPTY) >

<!ELEMENT anchor (EMPTY) >
<!ATTLIST anchor name %URL; #REQUIRED>

<!ELEMENT img (EMPTY) >
<!ATTLIST img src %URL; #REQUIRED>
```

A quite different use of markers is very common: mapping tabular data from a relational database into a set of empty markers enriched with attributes. In that case, the location of a marker indicates which table that entry belongs, while attributes indicate values in the original database. In the following example the elements `row` are markers collected by using the pattern *table*, I discuss later on:

```
<table>
 <row Name="Alice" Tel="01-54321"/>
 <row Name="Bob" Tel="01-12345"/>
</table>
```

3.2 Atoms

While markers are characterized by an empty content-model, users need a construct to mark-up units of information, unstructured and not further divisible. I refer this pattern as *atom*, to highlight the fact that only raw text can be included in its content model. An atom contains a sequence of characters, which express a basic content such as a date, a string or a number.

```
<!ELEMENT name (#PCDATA) >
<!ELEMENT place (#PCDATA) >
```

Atoms are mainly used in two scenarios: (i) within a text stream in order to capture the role of a fragment, for information retrieving, indexing and searching or (ii) as units of information collected in a more complex structure. Examples of the second approach are the elements `name`, `capital`, `population` in the following record:

```
<state>
        <name>Italy</name>
        <capital>Rome</capital>
        <population>56.000.000</population>
</state>
```

The relation between markers and atoms deserves a deeper explanation. From a syntactical point of view, a marker could be considered an atom without text. However, these two patterns are meant to play two different roles: the meaning of a marker relies on its location, while the meaning of an atom relies on its textual content. Schema designers should keep this distinction in mind and use each pattern for its straightforward purpose. Yet, some data (for instance, those extracted from a relational database) can be expressed as a record of empty markers with attributes, or as a record of atoms. Both of these solutions can be adopted (even if I admittedly prefer the second one): what is important is that they both are used consistently in the whole document, in order to increase readability and disambiguity of the document itself.

3.3 Blocks and Inline Elements

A mixed content model cannot be neglected discussing document patterns. Undoubtedly, the most common structure authors use are blocks containing text stream and unordered and repeated nested elements. The pattern *block* is used to capture paragraphs, titles, lines, verses, blockquotes and so on: whenever an author needs to combine text in a meaningful and, in a way, independent unit they need to use a block.

Note that patterns do not allow text to appear wherever in a document, but only wrapped by a block container. Even if a text fragment is apparently out of a paragraph (as allowed by HTML), it is conceptually wrapped and bounded by some structure. By making that structure explicit, and then by reducing the possibility of exceptions and variations, simpler and faster applications can be designed.

The concept of ”block” is actually inseparable from the concept of ”inlines”, i.e. the elements it contains, repeatable, in any order and mixed with the text. What makes a block different from an atom is just the presence of those elements. Even more, it is important the fact that no rules are imposed over their position within a block. It is the natural flow of the text, that determine the position of each inline structure. Furthermore it often happens that inline structure nest arbitrarily (as is the case of bold and italic elements). This means that in a descriptive environment it is hopeless and erroneous to try to impose any constraint on block elements except the complete identification of the allowable inline elements, that can nest arbitrarily.

Block elements and inline elements, thus, share the same content model, which is mixed and contains the list of the inline elements. Block elements are distinguishable because they use the same content model, but are not listed in the allowed elements. A simple way to express this is to employ a parameter entity used by both block and inline elements and not containing the block elements:

```
<!ENTITY % inline "(#PCDATA | cite | span | ... | b)*">
<!ELEMENT para %inline; >
<!ELEMENT blockquote %inline; >

<!ELEMENT cite %inline; >
<!ELEMENT span %inline; >
...
<!ELEMENT b %inline; >
```

3.4 Records

Positions, unstructured information and free text are not enough to cover all the possible situations. Designers need some more patterns to deal with structured information and to make

explicit the relations among elements, dependencies and repetitions.

The pattern *record* has been introduced to model all those circumstances where a limited set of elements need to be gathered under the same name or superstructure. More precisely, I refer to a *record* as a container of heterogeneous information, organized in a set of optional elements, non repeatable. Apart from non-repeatability, few rules are imposed over each element: a record cannot contain raw text, inlines elements or empty content-model, but any other pattern is allowed.

Note that a marker with multiple attributes can substitute for an entire record, as long as elements of the record can be represented fully by strings without markup. Two points are relevant to clarify their differences: that a marker is first of all characterized by its position, and that a record is also meant to model more complex structures and substructures (as fields with marked-up content).

Records can be first used to group simple units of information in more complex structures. In that scenario, each element is a pair *name-value* indicating a small piece of information about the whole object described by that record. The record `state` in the example describes a nation by reporting its name, capital and number of inhabitants.

```
<state>
        <name>Italy</name>
        <capital>Rome</capital>
        <population>56.000.000</population>
</state>
```

Furthermore, records are useful to organize data in hierarchical subsets. Each of these subsets can be arbitrarily complex or composed by arbitrary sub-structures (that, once again, follow patterns). In the example the element `book` is a record containing an atom (`title`), a record (`bookinfo`) and some elements named `toc` and `bookcontents`. In turn, the elements `bookinfo` is a record of an atom (`isbn`), a block(`legalnotice`) and `keywordset` which is a container of homogeneous elements `keyword`.

```
<!ELEMENT book (title, bookinfo?, toc?, bookcontents?)>
<!ELEMENT bookinfo (isbn?, legalnotice?, keywordset?)>
...

<book>
        <title>Prey</title>
        <bookinfo>
                <isbn>89312793-3213-afdsa-1<isbn>
                <legalnotice>Do not copy<legalnotice>
                <keywordset>
                        <keyword>prey</keyword>
                        <keyword>nanoparticles</keyword>
                <keywordset>
        </bookinfo>
...
</book>
```

A very important point is about the ”order” of the elements in a record. A record is a set of information, whose order is not relevant from a descriptive perspective. Designers are not interested in prescribing a priori *where* elements can appear; rather, in collecting a set of related information.

In the previous example I used the ’,’ operator to indicate a set of elements, since the fragment has been extracted from the XML DocBook specifications. A more descriptive declaration would be:

```
<!ELEMENT book (title & bookinfo? & toc? & bookcontents?)>
<!ELEMENT bookinfo (isbn? & legalnotice? & keywordset?)>
```

This declaration works for SGML documents. Similarly RelaxNG[Mur00] allows users to write a similar declaration for XML(the same DocBook specifications have recently moved to RelaxNG). XML DTDs lack the '&' operator and child elements are not allowed to occur in any order. The record pattern for descriptive XML DTDs deserves special attention: the order of the elements is not meant to be an imposition to make some documents invalid, but a non-meaningful order from a semantic point of view. The pattern in fact is meant to gather a set of related information without any specific order, but the XML DTDs do not make it directly possible. Note that the patterns discussed in this thesis are abstract guidelines, independent from specific schema languages.

Another factor characterizing a record is the non-repeatability of its elements. I haven't imposed a so strong limitation because repeatable elements are useless; on the contrary, I want to clearly distinguish repeatability by using a specific pattern for that (table). I was looking for a way to group repeatable elements into special containers, able to make explicit and unambiguous the functional relation among them. Consider, for instance, a book composed by a title and a list of chapters. I might have made the pattern record a bit more complex, by allowing repeatable elements too, in order to accept declarations as follows:

```
<!ELEMENT book (title, chapter*)>
<!ELEMENT chapter (para*)>
```

To maintain certainty and simplicity of patterns I have preferred to prevent those declarations and move the concept of repeatability in a different pattern.

3.5 Tables

A table is an ordered list of homogeneous elements. Tables can be used to group homogeneous objects into the same structure and, also, to represent repeating tabular data. In the following example `persons` is a table of records `person`, while `phones` is a table of atoms `phone`.

```
<persons>
        <person>
                <name>Alice</name>
                <phones>
                        <phone>02-2910830</phone>
                        <phone>02-8390211</phone>
                </phones>
        </person>
        <person>
                <name>Bob</name>
                <phones>
                        <phone>03-3271891</phone>
                        <phone>08-281038</phone>
                </phones>
        </person>
<persons>
```

Within tables users can expect to find atoms, blocks or records, but never inlines or markers. Inlines are not allowed because they are meant to appear only within a block; markers are not allowed because their position would be irrelevant in a superstructure where all elements are markers. A good way to emphasize its role as "set of homogeneous elements" is to name the table with the plural form of the name of the contained element.

```
...
<!ELEMENT persons (person)+ >
<!ELEMENT phones (phone)+ >
...
```

Tables are the main way for expressing repetitions. These repetitions are not expressed raw, as a subgroup, within a more complex content model, but protected by a plural-form wrapper that acts as a member of a more fundamental record. That is why I have distinguished tables and records, making both of them more restrictive and rigorous.

3.6 Containers

It is very common to have objects that need to be either repeated or collected under the same superstructure. In that case, records cannot be used because they are not supposed to model repeatability; on the other hand, tables are not suitable because of the required homogeneity of their content model.

Examples of such need can be found in many markup languages. The element `body` in HTML is a sequence of repeatable elements; the declaration of the element `div` in TEI; equally the element `bookinfo` in docbook and so on:

```
<!ENTITY % body.content "(%heading | %text | %block | ADDRESS)*">
<!ELEMENT BODY O O  %body.content>
```

I have then introduced a new pattern, called *container*, to cover all those situations. A container is an unordered sequence of repeatable and heterogeneous elements. The name emphasizes the genericity of this pattern, used to model all those circumstances where diversified objects are repeated and collected together.

As expected, the content-model of a container includes markers, atoms, blocks, records, tables and containers themselves. Only raw text and inlines (besides the empty content-model) are excluded, because they have to be wrapped within a block. It is no accident that recors and containers share the set of elements in their content-model. What changes is only the repeteability of those elements, since the order is not relevant in both cases.

Containers are clearly related with tables too, because of their repeatability. The only difference is that items of a containers are heterogeneous, while those within a table are homogeneous. From that perspective, a table could be considered a special class of container. However, I have preferred to distinguish them, to emphasize the difference between homogeneous and heterogeneous structures.

3.7 Additive and Subtractive Contexts

Not all situations designers find in descriptive markup can be covered by the previous patterns. Exceptions and special cases abound that can be dealt with difficulty with traditional validation languages, and easily with patterns.

For instance, one may consider allowing in an element other elements already used in other parts of a document, only with a few more elements not found elsewhere. One example is immediate: the `FORM` element of HTML allows all elements in the `&flow;` entity, plus the special form elements such as `INPUT`, `TEXTAREA`, etc. In a word, `FORM` provides a context for these elements. A context where a few elements are added in depth to existing elements is an *additive context*.

A different example regards re-using a content model already used in other parts of a document, only excluding some elements. Yet again, an example from HTML can be easy: `A` elements cannot contain other `A` elements. Similarly, users could define a `footnote` as a regular paragraph, except that no footnotes can be defined. Here again the `A` element and the `footnote`

element describe a context where some elements that would normally be allowed make no sense and should be signaled. That is a *subtractive context*.

```
<!ELEMENT contract %flow; +(signature)>
<!ELEMENT signature EMPTY>
```

The additive context and subtractive context patterns allow designers to explicitly express these relationships. Unfortunately, with traditional XML schema languages (DTDs and XML-Schema), it is very difficult to describe either additive or subtractive contexts: special elements can occur (or be excluded) not only directly within the container, but also within other elements inside it. On the contrary, RelaxNG, SGML's DTDs and languages for coconstraints such as Schematron[Jel05], SchemaPath[SCMV04] can adequately describe such situations.

These patterns add a lot of power and flexibility, as well as complexity, to the schemas and documents created by designers. For the purposes of my thesis, however, they can be set aside. My objective is designing a simple language able to capture *a posteriori* the structured content of any document, so I am not so much interested in the *conditional presence* of some elements (and actually it is difficult to decide whether or not we are looking at an additive/subtractive context or a simple content model) as on the only *presence* of them.

4 From descriptive to constructional

Traditional pattern-based approaches consist of identifying the most useful solutions in a given context, and reuse them. My thesis takes a different perspective: rather than limiting to identify and investigate those patterns, I suggest to only use exclusively them to write (and to segment) digital documents.

The basic idea is that *any* document can be *projected* into a strict composition of a limited set of objects (patterns), which express the same fundamental information of the original document in a simple, clear and unambiguous way. Once again, 'the same fundamental information' means 'the same structured content' according to the Pentaformat model and the premises of chapter 3. A document **identical** to the original one cannot be obtained by only using those patterns; but the content structures of that document can be captured without losing generality and applicability. Two orthogonal dimensions characterize such approach:

- *syntactical minimality*: the number of syntactical choices available for designers is extremely reduced. Note that minimality does not mean producing smaller schemas or documents, rather using a smaller set of syntactical choices.
- *semantic expressiveness*: pattern-based documents make explicit the semantics of structures, relations and dependencies.

4.1 Syntactical Minimality

Few objects and composition rules are available to express all the structures of a document. The property of syntactical minimality is justified from the *descriptive* nature of the schemas and documents we are dealing with. The examples of alternatives (which represent a relevant structure only to enforce a choice), repeatable subgroups (which should be surrounded by wrappers) or flow text (which should be wrapped by blocks) discussed in section 1 have shown how validation languages offer much more choices of structures than necessary in a descriptive environment.

In that context designers do not need to extend the set of available structures, in order to accommodate the plurality of situations. On the contrary, they can handle those situations by reducing structures to a limited set of constructs.

Table 4.1 summarizes the patterns discussed so far, highlighting which is the content model of each of them, in DTD syntax. What is evident from that table is the *orthogonality* of the patterns: each of them has a specific role and covers a specific situations, and no content model is

Pattern	DTD syntax
Marker	<!ELEMENT X EMPTY>
Atom	<!ELEMENT X (#PCDATA)>
Block	<!ELEMENT X (#PCDATA | E1 | ... | En | M1 | ... | Mn | Ax)*>
Inline	<!ELEMENT E1 (#PCDATA | E1 | ... | En | M1 | ... | Mn | Ax)*> ...
Record	<!ELEMENT X (E1?, E2?, ... , En?)>
Container	< !ELEMENT X (E1 | E2 | ... | En)*>
Table	< !ELEMENT X (E)*>

Table 1. Patterns and Content-models

repeated. Since a direct mapping exists between the most common needs of designers and these patterns, whenever a designer has to create a new schema he/she has to picture the current scenario and to select the only possible pattern that fits it. For instance, text fragments can appear only within blocks (inlines) or atoms, unstructured information can be carried only by atoms, homogeneous repeatable elements can be modeled only by tables, as well as containers serve whenever heterogeneous and repeatable elements are required, and so on.

Another important aspect of such pattern-based theory is that specific rules are imposed about which objects are allowed within which one. Table 4.1 shows these constraints, summarizing what I said about each single pattern (each row indicates elements allowed in the content model of each pattern).

	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

Table 2. Composition rules over patterns

Although it seems a limitation, such strictness contributes to widen the expressiveness and the applicability of patterns. By limiting the possible choices, in fact, the role played by each pattern is highly specialized and it is possible to associate a single pattern to the users' needs. For instance, preventing records within blocks we prevent an uncontrolled mixing of structured and unstructured content, or preventing inlines out of blocks we prevent incorrect locations for text fragments, or preventing tables within blocks we ensure the distinction between block texts and complex data structures, or allowing tables within records (and vice versa) we make possible the interaction of heterogeneous and homogeneous set of data, and so on.

While the limitations of atoms and markers are quite expected, it is worth spending some words about blocks and inlines. Blocks, in fact, cannot contain further blocks or more complex structures, contrary to what many markup languages say (for instance, DocBook allows `para` to contain `table`, `address`, `itemizedlist`, etc.). In my mind blocks are unified chunks of text, that cannot be further divided in sub-parts. Two cases are very common contrary to my position: embedded paragraphs and tables in paragraphs. Apart from this kind of tables which is not difficult to criticize as an example of bad-design, embedded paragraphs are very often used by authors, and allowed by specifications. I do not think a paragraph containing a paragraph is a structure that actually models the logical meaning of that fragment. On the contrary, I would model it by using either in-line (if the content follows the stream of text and belongs to the same speech) or by splitting it in three continuous paragraphs (if they are logically independent blocks). Whatever presentation is associated to that embedded fragment, what really matters is

the relation between that element and the block it is contained in.

Blocks play a central role, being the only place where text (mixed to in-lines) can appear. While atoms are used for unstructured information, blocks model all those situations where free text is written by authors. Note that patterns do not allow text to be directly used within containers (or tables, records and so on), but always wrapped by blocks. The rationale is a clear distinction between objects that express relations among elements, and objects that express the actual content (intended here as ultimate sequences of characters) of the authors. Once again, such strictness aims at making clear and unambiguous the role of patterns and their composition. Consider as example the following HTML fragment (remind that HTML allows users to insert text as a child of `td`, `li` or `div`) normalized to be patterns-compliant.

```
<ul>
<li><p>Paragraph in item 1</p></li>
<li><p>Paragraph in item 1</p></li>
<li>
 <p>Paragraph in item 1, followed by a table</p>
 <table>
  <tr>
   <td><p>A</p></td>
   <td><p>B</p></td>
  </tr>
  <tr>
   <td><p>C</p></td>
   <td><p>D</p></td>
  </tr>
 </table>
</li>
</ul>
```

The tables `table` and `ul` provide the macro-organization of content; containers `li` and `td` wrap each item of the same cluster; elements `p` wrap the actual information. Then, patterns sacrifice the compactness of documents but gain readability and disambiguity.

Table 4.1 remarks the similarity between records, tables and containers. They are all meant to describe the logical organization of document's fragments, and to express hierarchies within the document. For this reason, they can contain further patterns except text and in-lines, which always need to be wrapped by a block. What changes is only the repeatability and optionality of their content elements. A clarification is needed about tables: since they are meant to gather homogeneous elements, it does not make much sense a table containing a sequence of milestones, markers whose information is the position in the document, while equipped markers are allowed (see section 3.1 for details about their distinction).

Then a strictly pattern-based schema (or a DTD) contains only seven patterns, enriched with some attributes, and composed according to these rules. The whole schema is very simple and easy to be read. The question is rather whether or not all the possible structures of digital documents can be covered by a so small set of objects. Two considerations answer that question. First of all, a reminder about the nature of the schemas (documents) we are looking for: patterns do not allow to write *any* schema, but to write *any descriptive schema for structured content*. The fact remains that they are good practices for designing well-structured, clear and reusable documents too. The second point is about the role played by the wrappers. Most of the existing schemas are not natively consistent with patterns, and rely on declarations where alternatives, repeatable elements and optional ones are mixed together. Those declarations can be normalized by introducing some wrappers, which actually make *syntactical minimality* possible.

Basically wrappers "spread" the meta-information over the depth of the document in order to decrease the need for complex constructs. Consider, for instance, the element `section` in the DocBook DTD. From a descriptive point of view, that declarations suffers two main issues: first, it expresses some constraints that can be omitted in an *a posteriori* analysis (it prevents to have

subtitles without titles, it prevents to have `refentry` followed by paragraphs, it imposes that a bibliography, if exist, is located at the beginning or at the end of the section, and so on); second, it assumes but hides the existence of conceptual containers for similar objects (for instance, the equipments `toc`, `index`, `glossary`, as well as the block-level elements like `beginpage`, `para`, etc.). Even visually that declaration is quite complex and difficult to be interpreted.

```
section ::=
(sectioninfo?,
 (title,subtitle?,titleabbrev?),
 (toc|lot|index|glossary|bibliography)*,
 (((calloutlist| ... |para|simpara| ...
    address|blockquote|graphic|graphicco|mediaobject| ...
    informaltable|equation|example|figure|table| ... |beginpage)+,
   ((refentry)*|
    (section)*|
    simplesect*))|
  (refentry)+|
  (section)+|
  simplesect+),
 (toc|lot|index|glossary|bibliography)*)
```

The following fragment shows a radical and pattern-based simplification of that declaration, based on a methodical use of wrappers:

```
section        ::= (sectioninfo?, wr:titles?, wr:equipments?,
                    wr:contents?, wr:subsections?, wr:equipments?)

wr:titles      ::= (title?,subtitle?,titleabbrev?)
wr:equipments  ::= (toc|lot|index|glossary|bibliography)*
wr:contents    ::= (calloutlist| ... | ... |beginpage)+
wr:subsections ::= ( refentry | section | simplesect)*
```

Titles are collected under a wrapper `wr:titles` (allow me to use different namespaces in a DTD, as proposed by Amorosi et al. with DTD++[AGV03]), which is a record of specific titles. Once again, the fact that subtitles are allowed without a title is not relevant here. Equipments are collected in a wrapper/container `wr:equipments` that makes explicit their belonging to the same category, as the element `wr:contents` does for the text blocks like paragraphs, quotations and so on.

Particularly meaningful are the transformations about `refentry`, `section` and `simplesect`. The DocBook DTD prescribes these elements, if exist, are located after text blocks; moreover only one of them can actually be present. Since the information they provide is somehow unifiable, I have added a wrapper `wr:subsections` which acts as a general container for them. The constraints about their mutual exclusion has been relaxed, assuming we are in a pure descriptive environment.

Actually such a declaration loses another relevant information: the homogeneity of repeated `refentry`, `section` and `simplesect`. A different pattern-based solution consists of creating one wrapper for each of them, in particular a table in plural form, and explicitly inserting these new wrappers in the content-model of the record `section`. Although it apparently imposes an order among elements, this solution does allow users to write documents which express the same structured information of the original ones:

```
section        ::= (sectioninfo?, wr:titles?, wr:equipments?,
```

```
                    wr:contents?, wr:refentries?, wr:sections?,
                    wr:simplesects?, wr:equipments?)

wr:refentries  ::= (refentry)*
wr:sections    ::= (section)*
wr:simplesects ::= (simplesect)*
```

The point is that these pattern-based declarations do not have the same role of the original one, and do not want to express the same constraints. What they want to do is describing from an higher perspective the whole set of documents validated by the original definition.

An evident drawback of using wrappers is the verbosity of the schemas. Such verbosity is balanced by the clearness and disambiguity of the resulting schemas. The property of *syntactical minimality* in fact does not imply the minimality of the schemas' dimension or documents complexity, rather the reduction of the number of choices available for designers.

Some interesting works in the literature anticipated and discussed the need of minimality: Usdin[Usd02] claims that designers are interested in flexible semantics and not in flexible syntax, observing that, if different people might produce different, but correct, documents to express the same meaning, the risk of misinterpretation is increasing. Patterns severely limit the choices in structures and composition of elements, while maintaining full descriptive in the definition of elements and attributes. Thus my approach agrees with Usdin's point about limiting the flexibility of syntax. What patterns propose is not 'syntactic sugar', but rather a limited, well-defined and understandable set of meaningful choices: errors and misunderstandings are then minimized by minimizing the choices.

4.2 Semantic expressiveness

The strength of a pattern-based approach is that syntactical minimality does not imply loss of expressiveness, but it is a vehicle to create well-structured, unambiguous and manageable schemas. In particular, two properties of patterns are very important:

- *Strictness*: each pattern has a specific goal and fits a specific context. The orthogonality between content models make possible to associate one single pattern to each of the most common situations in document design. Then, whenever a designer has a particular need he/she has to only select the corresponding pattern and to apply it.
- *Assembly*: each pattern has a specific content-model and can be used only in some locations. The composition rules of patterns ensure a strong separation between objects that capture the logical organization of the document, and objects that actually carry the ultimate content.

These two properties make a pattern-based language a good solution to highlight functional dependencies and semantics of document components. In particular, patterns make explicit the semantics of structures, relations and dependencies by using *wrappers*. The role of wrappers has already been discussed in the previous pages, as means for *syntactical minimality*: by introducing wrappers, in fact, any structured content can be normalized as a combination of very few objects according to a very small set of rules. Here the focus is about the expressiveness of wrappers and their application to disambiguate information.

Consider for instance the following declaration, clearly contrary to the patterns discussed above:

```
<!ELEMENT T1 (K, (B|C)*)>
```

The content model of `T1` suggests a tacit relation between the element `K` and the element `B` and `C`, or better a relation between `K` and repeatable alternatives of `B` and `C`. Probably `K` is a unique information, that identifies something in the schema (imagine a key in a database). That information is hidden in the schema, but can be explicited by adding a wrapper `WR`.

```
<!ELEMENT T1 (K, WR)>
<!ELEMENT WR (B|C)*)>
```

Repeatable sequence of elements are very common in document design, but they are another example of declarations that can be improved by patterns.

```
<!ELEMENT T1 (B,C)*>
```

The most natural reason to write such content model is the need of imposing a predefined order between pairs of elements `B` and `C` (for instance pairs question/answer) and not allowing them to appear in isolation. That logical connection can be expressed again by adding a wrapper `WR`.

```
<!ELEMENT T1 WR*>
<!ELEMENT WR (B,C)>
```

Wrappers are useful to clearly define the scope of some elements, as well. Consider for instance the following XML fragment:

```
<title>Example Title</title>
<para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
<para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
<para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
<title>Example Title</title>
<para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
<para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
```

Without wrappers readers cannot say whether or not the first `title` is the title of the whole document or the first section; moreover they cannot say whether the second one is at the same level or is a subsection title. Yet, attributes can be used but they require an explicit computation to rebuild on-the-fly the actual logical structure of the document. By introducing some wrappers that disambiguity disappears (I show only an example but it is simple to imagine how the document looks like, with different organizations):

```
<title>Example Title</title>
<content>
 <para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
 <para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
 <para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
 <section>
  <title>Example Title</title>
  <content>
   <para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
   <para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.</para>
  </content>
 </section>
</content>
```

The three simple examples I provided, and those discussed in the previous section, show how wrappers can be used to express the semantics of documents, in order to make information interpretable either by humans and by machines. The semantics of metamarkup languages such as

SGML and XML has been widely studied in the literature. Usdin[Usd02] raised a fundamental question: can the users infer something that authors had not implied? What a document says is always what the author really would say? Raymond et al.[RTW96] and Sperberg-McQueen et al.[SMHR00] remark that an XML document (but this also applies to SGML) need some extra information to be interpreted by humans, in particular names carefully selected by domain experts. Thus, of itself, XML is only partially suitable to interchange information among machines: while humans have a common ontology (the word 'title' indicates something that is a 'title'), machines do need a common and unambiguous semantics of the same tags. More recently, Renear et al.[RDSM02] discussed the importance of such a clear semantics describing the BECHAMEL Markup Semantics Project, a system for expressing semantic rules and meanings for markup languages based on PROLOG inferences and deductions[DMHR03]. The relation between BECHAMEL and the Semantic Web [BLHL01] is evident (both of them want to transform information in something completely interpretable by machines): but while the latter looks at these issues from a more general perspective, BECHAMEL focuses the attention on a specific domain.

While BECHAMEL and related works build a metastructure that can infer rules and semantics from the language, my patterns have a different goal: proposing a restricted set of structures and substructures that already have intrinsic and unambiguous semantics. Yet, patterns are not concerned with the semantics of names and objects, but rather with the semantics of structures and the relations and dependencies among the elements.

4.3 Evaluating the Pattern-based approach

To conclude, it is interesting to compare my pattern-based approach with some positions discussed in chapter 2. First of all, my approach is in line with prior research on markup about trees (see 1.3 for details), which said they actually capture relations and dependencies between document parts. I totally agree hierarchies allow users to minimize redundancy, localize dependencies and increase expressiveness since they allow authors to identify functional dependencies among data, regardless of the nature of those data. For this reason, patterns basically add "depth" to the documents, accepting some extra verbosity.

Classifying a pattern-based markup language according to the taxonomies analyzed in 1.1 is another point of interest. First, note that my model is not a specific XML dialect, but a metalanguage: it is not a single block in the taxonomies proposed by Piez, Renear and Wilmott, but an area covering different blocks which address different domains. All the languages in that area are based on the same design principles. I would label my approach (or, better, any language derived from it) as *retrospective* and *metaleptic*: retrospective because a pattern-based schema models existing data from a general perspective without imposing strong constraints; metaleptic because the simplified usage of patterns makes efficient and reliable the future management of the same data. Taking into account similarities and partial overlap between Piez's and Renear's classifications, pattern-based languages have an *indicative* mood in a *logical* domain. It also worth to investigate whether or not patterns generate *exploratory/mimetic* languages: more than 'exploratory' as Piez meant (patterns are not adaptable to the exceptions and irregularities as required), patterns can be defined 'mimetic' because they allow users to create schemas and instances that can be almost blurred each other. That is possible because of their non-ambiguity and strictness.

Again, according to Wilmott's classification (markup languages for humans or machines), a pattern-based approach cannot be placed at the extreme of the spectrum: it is *human-based* because of the idea of noise minimization, readability, minimization of constructs that may appeal to human readers, but also *machine-based* because of its ease of processing by future applications.

The path along this chapter should have made clear the nature and objectives of markup languages based on the patterns discussed so far. Patterns are 'descriptive' and strictly connected with the Pentaformat model presented in 2. They are meant to express the structured content of any document, in a clear and unambiguous way. A first application is then for a *segmentation model* adopted by designers who need to extract information from legacy documents, and build applications that manage that information. Moreover, the reliability and re-usability of patterns (see section 2 for details) joined with their semantic expressiveness and syntactical minimality

(see section 4) make them a good solution to create new documents. They can also be used for a *constructive model* adopted by designers who want to create well-structured, clear and reusable documents from scratch.

Chapter 5

A Pattern-based Minimal Language: IML

The patterns presented in the previous chapter encode two kinds of information: some classes of elements widely used and enough powerful to capture the most natural structures of a document and some rules to put them together in a straightforward and unambiguous manner. No information is provided about the actual instances in each class, no name is imposed, no set of attributes suggested. They are "composition rules" and best-practices that should be followed in order to create simple, unambiguous and modular documents. By applying those patterns we can then design different markup languages for different domains.

In particular, they are meant to be used for the design of an *abstract* language which expresses the structured content of *any* segmented document, according to the Pentaformat model. In this chapter I present that language, called IML, discussing its relation with the models and patterns proposed so far.

1 From abstract patterns to IML

Assuming that patterns discussed so far are versatile enough to cover the most common situations, the process itself of designing a markup language is heavily simplified. What designers have to do is simply deciding which are the elements in each category (pattern) and which are their properties. The content model of most of them will be automatically derived from their classification.

In a context where choices are reduced by the presence of patterns, understanding what and how many elements belong to each category is a step to be alike weighted. Choosing between a verbose and detailed language, or a small and general one has a great impact on future applications. Two main philosophies support that decision:

- an *exhaustive* approach: which uses specialized elements, with specific names and roles. The meaning of each object is captured by its name, based on an *a priori* classification of the constructs. Examples are the specifications of TEI[Con87], DocBook[Wal99].
- a *minimalist* approach: which uses a very small set of constructs and characterizes objects by using attributes and properties. The meaning of each object is captured by the category it belongs to and the value of some attributes.

Consider, for instance, an XML fragment to model a book, divided in chapters, each composed by paragraphs and quotations. Block elements contain footnotes, italic fragments and in-line elements to mark-up places. The following fragment adopts an exhaustive approach.

```
<document>
<introduction>

<para>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.
Lorem Ipsum. <footnote>Footnote</footnote> Lorem Ipsum. </para>

<blockquote>Citation. Citation. <place>Place</place>
Citation. Citation. Citation. Citation. </bockquote>

</introduction>
</document>
```

The same document can be modeled in a minimalist way, as follows:

```
<div class='document'>
<div class='introduction'>

<p>Lorem Ipsum. Lorem Ipsum. Lorem Ipsum. Lorem Ipsum.
Lorem Ipsum. <span class='footnote'>Footnote</span>
Lorem Ipsum. </p>

<p class='blockquote'>Citation. Citation. Citation.
Citation. <span class='place'>Place</span>
Citation. Citation. Citation. Citation. </p>

</div>
</div>
```

No approach is absolutely better than the other, but designers can (and actually do) choose one of them (or hybrid solutions) according to their needs and preferences. However a *minimalist* approach is the right choice for a language expressing segmented content. Once again, the point is that we are not looking for a language able to express constraints, to restrict elements' occurrences, to prevent errors and to *prescribe* specific rules but a general description of the structured content. A minimalistic approach would be more difficult to be mastered in prescriptive schemas: the validation of element occurrences based on their attribute values, in fact, is not possible in many schema languages (for instance, XML-Schema and DTD do not handle co-constraints while RelaxNG and SchemaPath do). On the other hand, many reasons make such approach a good solution for a descriptive language. First of all, because that language should be able to capture the meaning of *any* document, regardless of its format and subject. It cannot use specific tag names and attributes, but a set of flexible and customizable objects. Second, because that language does not express *a priori* constraints over content and defines specific content-models, but rather identifies *a posteriori* basic objects and their roles. A third motivation is related to the purpose of this work, which aims at supporting automatic conversion: such conversion can be generalized and implemented with less effort if the input language is minimal and rigorous.

For that matter, a minimalist approach derives directly from the pattern-based solution proposed in the previous chapter. Assuming that few patterns are enough to express *any* content, a language based on those patterns has nothing to do but say which pattern each object respects (e.g., whether the object is a block text, a container, a table or an inline) and which specific class it belongs to (which kind of blocks it is, which level of nesting it has, and so on).

1.1 Extreme IML

Extreme IML is an experimental language to examine the nature of IML and its relationships with the theory of patterns discussed so far. From section 4, we can derive a simple method to summarize pattern-based schemas (or DTDs): simply by indicating in a table the names of the

Pattern	Elements	Content Model
Markers	span	EMPTY
Atoms	span	#PCDATA
Blocks	p	(#PCDATA | %Inlines;)*
Inlines	span	(#PCDATA | %Inlines;)*
Records	div	(div)*
Containers	div	(div | %Tables; | %Blocks;)*
Tables	div	(div)*

Table 1. Extreme IML

objects belonging to each class (pattern). Since each pattern has a rigorous content-model, we can easily derive the content model of each element. Few more information are needed to specify properties and attributes, but the overall organization of the document is clear.

Table 1.1 adopts that method and shows ExtremeIML, an actual language, where only one single element belongs to each category. Three elements, expressed for instance in XHTML syntax, are enough to map all patterns: generic element `P` for text blocks, a generic element `SPAN` for all kind of objects possible in a block (so inlines, markers and atoms), and a generic element `DIV` for containers and tables. Note that a record is expressed as a sequence of `DIV` diversified by attributes. What is further needed is a way to characterize instances, for example to differentiate a paragraph of normal text from a title or a list from a table, and so on. The attribute `@class` is a natural candidate to do that. The ExtremeIML DTD is very simple; I would even say 'embarrassing':

```
<!ENTITY inlines "(#PCDATA | span)*">
<!ENTITY attrs "class CDATA IMPLIED">
<!ELEMENT div (div | p)*>
<!ATTLIST div
%attrs;
>
<!ELEMENT p %inlines;>
<!ATTLIST p
%attrs;
>
<!ELEMENT span %inlines;>
<!ATTLIST span
%attrs;
>
```

A so simple schema is ideally enough to model (the content of) any document, i.e to encode a document normalized to the patterns. The following fragment, for instance, shows a representation of a document with a list, a table, some paragraphs and some fragments in italic and bold. Although a bit 'naive', such representation captures the same meaning of an exhaustive one, that uses tags like `table`, `ul`, or `italic`.

```
<body>
 <p class='title'>Title</p>
 <p class='normal'>Normal paragraph</p>
 <p>Normal paragraph with <span class='italic'>italic</span>
        and <span class='bold'>bold</span> inlines</p>
 <div class='list'>
  <div class='list-item'><p>Text in item 1.</p></div>
        <div class='list-item'><p>Text in item 2.</p></div>
```

```
 </div>
 <p class='text'>Normal paragraph</p>
</body>
```

Another important point is the syntax. The choice of XHTML is completely arbitrary and whatever set of names would be equivalent, since the relevant information are patterns, rather than actual tag names. I have chosen such a syntax because it is well-known, clear and directly representable in a browser (adding few CSS rules, the example can be rendered with actual tables and lists).

The strength of Extreme IML (and then IML) is its *generalization*. The innovation does not rely on names and attributes (which are really ordinary) rather on the fact that a so small set of objects, and pattern-based composition rules, models any document, and makes possible the implementation of advanced conversions. Extreme IML and IML are in fact characterized by two properties:

- *minimality*: the language is comprised by a very small set of elements, supplemented by some meaningful attributes.
- *rigour*: each element follows a specific pattern, and elements are nested each other according to a restricted set of composition rules.

1.2 IML: a (not so surprisingly) simple DTD

IML can be seen as a 'reasonable extension of Extreme IML'. Some specialized elements are added for each category (pattern), in order to express the most common objects of digital documents, and to make their management more direct, simple and clear. IML is then a simple markup language composed by a very small set of tags with pattern-based content models, supplemented by some `@class` attributes. Table 1.2 shows these elements. I prefer such visualization to highlight the strict relation between patterns and IML, and its simplicity. It is not difficult to picture a more familiar DTD or Schema.

Pattern	Elements	Content Model
Markers	`img`	EMPTY
Atoms	`span`	#PCDATA
Blocks	`p`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`	(#PCDATA | %Inlines;)*
Inlines	`a`, `span`, `sub`, `sup`, `i`, `b`	(#PCDATA | %Inlines;)*
*Records**	`table`	(`tr`)*
Containers	`body`	(`div` | %Tables; | %Blocks;)*
	`div`	(`div` | %Tables; | %Blocks;)*
	`li`	(`div` | %Tables; | %Blocks;)*
	`td`	(`div` | %Tables; | %Blocks;)*
	`th`	(`div` | %Tables; | %Blocks;)*
	`tr`	(`th` | `td`)*
Tables	`ul`	(`li`)*
	`table`	(`tr`)*

Table 2. IML Core

As patterns capture the most used structures in digital documents, IML translates those abstract structures into an actual markup language. IML documents are simply a sequence of content objects that simply specify which pattern each object respects (e.g., whether the object is a block text, a container, a table or an inline) and which specific class it belongs to (which kind of blocks it is, which level of nesting it has, and so on) through the attribute `@class` and few more attributes for specific needs.

Consider for instance an "important" paragraph. Each format has a different way to express that information, but it is clear that a paragraph when style is "important" in MS Word, a fragment `<p class='important'> An important paragraph </p>` in HTML, and the fragment `<important> An important paragraph </important>` in XML are all conceptually equivalent. Similarly a text fragment rendered in italic because of its structural and semantic meaning is wrapped into an `i` element in IML, but it ends to be an italic inline info a PDF, an element `emph` for DocBook, an element `i` or `em` in HTML, a command *textit*{} in LaTeX, and so on. In both cases (but many others could have been cited) IML expresses that semantic meaning in a simil-XHTML syntax.

IML is indeed characterized by *syntactical equivalences*. Consider again the `i` element: it actually represents an italic fragment regardless of its syntax. Many syntaxes are then considered equivalent: `<span class='italic'> Italic </span>`, `<span class='corsivo'> Italic </span>` (in italian), `<em> Italic </em>`, and so on. Similarly a paragraph without any specific role can be expressed as `<p> normal paragraph </p>`, or `<p class='normal'> normal paragraph </p>`, or `<p class='MsoNormal'> normal </p>` (in MS Word), or as `<p class='BodyText'> normal paragraph </p>`, and so on. All these equivalences are encoded within IML-based converters (discussed in the next chapters), so that authors can label content as they prefer, and that content is normalized into a clear and processable IML representation.

IML is then very simple. The most important feature is the methodical use of the attribute `@class` to guarantee generalization and applicability to different domains. As I said, the innovation does not rely on tag names and attributes, rather on their *minimality* and *rigour* which made possible the implementation of automatic conversion between heterogeneous data formats. The following fragment shows an example of IML document:

```
<iml>
 <head> ... </head>
 <body>
  <h1>Main title</h1>
  <p class='blockquote'>Citation.</p>
  <p>Normal paragraph</p>
  <p class='important'>Important paragraph with some
  <span class='names'>names</span> and <i>italic fragments</i>.
  </p>
  <table>
   <tr>
    <td><p class='normal'>Para in a table.</p></td>
    <td>
     <ul>
      <li><p>Para in a list, in a table.</p></li>
      <li><p>Para in a list, in a table with a meaningful
      <span class='ref'>inline</span></p></li>
     </ul>
    </td>
   </tr>
  </table>
  <p class='caption'>Table caption</p>
 </body>
</iml>
```

The three main features of IML are evident from the example: first, IML describes only structured content and neglects presentation, behavior and metadata (the element `head` is temporary included in the documents, but still need to be deeply studied and used); second, IML strictly adopts the patterns (note for instance the mandatory presence of paragraphs around text, as well as the rigorous nesting of paragraphs and containers, and so on); third, IML makes intensive use

of the attribute `@class` to express the semantic role of content fragments.

What further characterizes IML is the presence of specific elements like `b`, `h1`, `ul`, etc. The choice of these elements is completely **arbitrary** and it depends on the fact that they are very common among authors and designers. Note that some elements are actually redundant and could have been omitted. IML keeps them because their wide use and support in documents' applications. Consider, for instance, the numbered headers `Hn`: they are probably not needed when `div` structures are nested correctly, but they are still included in IML because they can be very often found in digital documents. IML in fact can be used to model both plain and hierarchical documents.

The meaning and use of each element does not require further explanations: element `IMG` for images (equipped markers), `P` and `Hn` for paragraphs (blocks), `A` for links, `SUP` and `SUB` for superscripts and subscript, `UL` and `LI` for lists, and so on. The only element that requires a deeper discussion is the element `TABLE` used for records: since records model structured information, usually expressed as pairs *name-value*, IML represents that structures with a table of two columns, which indicate the two fields of the record.

```
<person>
 <name>Angelo</name>
 <surname>Di Iorio</surname>
 <telnumber>0009-00219091</telnumber>
<person>

<table class='person' IMLtype='record'>
 <tr>
  <th><p>Name</p></th>
  <td><p>Angelo</p></td>
 <tr>
 <tr>
  <th><p>Surname</p></th>
  <td><p>Di Iorio</p></td>
 <tr>
  <tr>
  <th><p>Telnumber</p></th>
  <td><p>0009-00219091</p></td>
 <tr>
</table>
```

Even if such reduction seems to be a bit convoluted, it allows users to express the relation between names and values of a record entry and, above all, to model any kind of record without requiring specific tag names or constructs. Note that records can be nested arbitrarily, since `td` is a generic container, whose content-model permits new records.

2 Merits and limits of IML

I expect at least two questions about the set of tags I propose: (i) why these tags, instead of others? (ii) how can users express objects which are not present in this list? To answer these questions, I propose a drawn-in-the-round evaluation of IML and a comparison with some well-known markup languages.

2.1 A meaningful language?

My claim is not only that arbitrarily complex documents can be written in IML, but even that existing ones can be normalized into simplified versions which use only such a limited set of structural objects, and still express the same content. The two abovementioned questions can be then paraphrased as follows: 'Is the minimality of IML enough to express *everything*'? The

answer is that IML is not meant to be the ultimate language that directly models *everything*, but a *core* language that can be customized for specific domains. I have chosen those tags because they are very common, and my personal experience has shown they are enough versatile to capture most of the documents. There is no reason to prevent for instance `acronym`, `dd` (or any other element) to be included in IML. Nonetheless remind that they can be expressed with an element `span`, enriched with attributes. Similarly, generic constructs like `p` and `div` (more `@class`) can be used to markup any kind of content. What is important, however, is the strict adherence to patterns.

Yet, some scenarios cannot be *directly* modeled with a so simple schema such as mathematical formulas, or graphical fragments, of forms, or fragments written in domain-specific syntaxes and so on. IML does not directly address them but can be easily extended. Adding some attributes and tags the whole structure of the document does not change, and the basic pattern-based constructs remain unchanged. What change are only some *local* names and components. Once again, the key-aspect of IML is not its *syntax* or *names' semantics*, rather the *minimality* and *rigour* behind the language.

A second relevant point is that IML is a natural consequence of Pentaformat and patterns, and it makes little sense without those premises. Let us think back to the path of the last three chapters: chapter 3 proved that a digital document can be divided in some segments, and highlighted differences and relations among those dimensions; then, chapter 4 explained how descriptive structures can be shrunk to a small set of objects, and how those objects capture the most relevant *content* (note once again the relation with the Pentaformat model) of *any* document; finally, IML instantiated abstract patterns into an actual markup language, readable and processable either by humans or machines. Section 2 described advantages of a pattern-based approach in terms of re-usability, flexibility and reliability. The same considerations are obviously extended to IML, that simply translates abstract patterns into tangible objects of a markup language.

The fact that both humans and machines can easily interpret IML is another essential feature. Wilmott[Wil02] discussed requirements for these classes of languages. From both sides, some benefits can be found in IML too: (i) *clear distinction between data and markup* derived from the XML syntax of the language, as well as (ii) *convenience for transmission*, (iii) *minimization of noise and normalization*, derived from the strict adoption of patterns and their expressiveness, (iv) *limited support meaningless variations*, derived from the descriptive nature and strictness of patterns, and (v) high *variability and flexibility*, derived from the use of `@class` and the generalization of constructs.

Another feature that is worth being remarked is the XHTML syntax of IML, that makes it directly readable by a browser. IML is then very close to the users, for two reasons: first, because an IML document represents the most common structures used by authors; second, because that document can be easily read and converted in other formats. Although some training/learning is required, in fact, IML is not difficult to be mastered by users.

2.2 A comparison with micro-formats

A very close solution to IML's approach are microformats, which embed semantic information within ordinary XHTML, by using few tags and attributes. The official microformats' web site describes them as "a set of simple, open data formats built upon existing and widely adopted standards"[CTK+05], and focuses on the fact they aim at solving simpler problems rather than creating a huge and complex semantic structure.

Microformats are not completely new. The SGML community have already proposed similar ideas many years ago, with the architectural forms of HyTime. For instance, DeRose and Durand[DD94] discussed the features of HyTime, a markup language introduced in the late 80s, which defines a set of hypertext-oriented element types that, in effect, allow SGML document authors to build hypertext and multimedia presentations. Basically, HyTime is a general hyperlinking and location addressing architecture based on a strong distinction between identifying typed objects, and using those identifications to express relations among them.

Microformats basically consist of a clever and rigorous application of some XHTML features, including the powerful `class` attribute, in order to describe places, people, events and so on,

and their relative relationships. This is a first interesting point of contact with IML, whose goal is expressing meaningful information *embedded* within a text by using very common objects. A relevant difference is about the names of those objects: unlike micro-formats, which use specialized tags for a specific context and have a pre-defined and rich set of names, however, IML adopts a flexible mechanism which can be used to model any content. In a sense, IML can be seen as a meta-microformat, that is a general schema from which specific microformat can be easily derived.

The management of semantic information is another perspective to compare IML and microformats. Microformats have been defined as a "lower-case semantic web", since they do not offer the rigor and soundness of standards like RDF and OWL, but they provide a bottom-up approach to add semantic information to the documents. That bottom-up approach has been one of the most important reasons of their success, joined with their easiness and specific applicability. Yet, the (uppercase) Semantic Web[BLHL01] is a complete framework which makes information fully processable by machines but it requires many effort to guarantee consistency, automatic reasoning and so on. I agree with the supporters of microformats, that consider them as an intermediate solution which can actually 'reduce the gap between users and semantics'. IML aims at describing semantics of document as well: while microformats are primarily concerned with the semantics of names and objects, however, IML primarily concerns with the semantics of structures and the relations/dependencies among elements.

Khare[Kha06] took a very interesting look at microformats, highlighting general principles behind them and discussing the phenomena of communities growing around these simple specifications. He outlined some benefits of microformat's approach, arguing against people who consider them only an *h* effect* (making fun of the 'h' prefix of most microformats). The same benefits can be found in IML, with some minor differences:

- *reduce*: microformats focus on specific problems and favor the simplest solution. IML adopts the simplest and most common solution to express what an objects is and how it is related with others. While minimality for microformats means a reduction of possible tags and names to handle a specific issue, for IML it means a reduction of constructs to handle different issues.

- *reuse*: microformats use existing standards and do no reinvent the wheel. IML does the same expressing the patterns behind the language in a very common and widely supported syntax.

- *recycle*: microformats allows users to recycle the same fragment in blog posts, Atom, RSS feeds and so on. IML fragments use an XHTML syntax and can be then easily moved from a format to another, from an application to another.

- *presentable and parsable*: microformats carry information visible and embedded within documents. An IML document can be interpreted by a browser, rendered with few CSS rules, processed by any HTML application as well.

While Khare expressed a totally positive opinion, Quin[Qui06] concluded that microformats are either 'good ingredients' or 'contaminants' for documents. Basically, he argued that they have a positive effect because they add a bit of semantic of a language mostly designed for presentation:web authors can deal with something more and web agents (machines) can work on a more structured and processable information. On the other hand, Quin considered microformats as contaminants since they force semantics to be expressed by something which is not meant to do that (actually he used the term 'subversive semantics'); microformats paradoxically help users to avoid using XML, but even HTML in its original acceptation. The discussion of Quin about advantages and disadvantages of microformats is helpful to further compare them to IML. Some points are relevant:

- *microformats are decentralised, simple and displayed by browser*: the fact that IML is a subset of XHTML makes it easy to create and render an IML document. Then, authors can di-

rectly write them or extract them from legacy resources. The minimality and simplicity of microformats apply to IML as well.

- *microformats generate name conflicts*: since microformats do not have global identifiers or something equivalent to namespaces, soon or later issues about conflicting and scalability (of an hypothetical server) will be arisen. IML does not aim at becoming a standard used for specific contexts, but an internal language for automatic analysis and conversion; the presence of conflicts it is not relevant from that perspective.
- *microformats generate documents hard to validate*: the fact that microformats use the attribute *@class* makes validation more difficult (unless using specific languages for co-constraints). IML is a descriptive language, not meant to avoid errors and prevent invalid schemas. Thus, difficulties of validation are less important now.

The conclusion of Quin's paper is a list of questions about the relationship between XML, HTML, the Web and the Semantic Web. Similarly the semantic role of IML (intended here as microformats do) still need to be clarified. Future works on the Pentaformat model, and investigations about the relation between IML, metadata and behavior will move such discussion forward.

2.3 A comparison with TEI and DocBook

The minimality of IML seems to contradict what most markup languages say, and to consider them too much complex and convoluted. IML does not aim at discrediting existing markup languages, but at abstracting and describing their structures. A comparison between IML and some common languages (TEI[Con87] and DocBook[Wal99]) is then useful to better understand my proposal. TEI is a standard that enables libraries, publishers, and scholars to represent a variety of literary and linguistic texts for online research, teaching, and preservation. Characterized by a huge amount of elements and tags (that can be further customized or extended), it allows encoders to work on any kind of text and to mark-up content with great precision. DocBook is a standard initially developed to encode scientific books and papers and then used to mark-up different texts. Based on a hierarchical model, it provides users a lot of constructs to model text, equipments, metadata and it has been widely used for documentation, books and personal encodings. The most relevant points of divergence between TEI (I will refer to TEI but similar considerations can be applied to DocBook as well) and IML are listed below:

- *prescriptiveness*: many aspects make TEI a prescriptive language, while IML is primarily descriptive. Section 1.1 defined a prescriptive language as a language that imposes rules over the structures of a document, and prevents forbidden encoding. Although it is meant to describe legacy resources *a posteriori*, TEI expresses many complex rules over structures and has many declarations which indicate content-models with fine-grained precision and variation. IML is a more general schema aiming at *abstracting* logical structures and relations.
- *descriptiveness*: dealing with the descriptive aspects of TEI, it is evident that TEI describes many aspects different from IML. It aims at describing each (small) detail of the original document in order to faithfully reproduce it. IML aims at extracting the most relevant structures and at normalizing them in processable objects.
- *naming and generalization*: TEI uses specific names and attributes, to create a digital copy of a non-digital resource, while IML uses a general schema, to capture (and, later, to characterize them) structures. As discussed at the beginning of this chapter, they respectively implement an *exhaustive* approach against a *minimal* one. The same dimensions of their DTDs explain such substantial difference.
- *scope*: TEI is a complete language designed to directly encode *all relevant* information, even if it can be customized for specific needs. IML is a *core* language (in a sense, a metalanguage) that can be adapted and modeled on specific domains.

- *multiple structures support*: TEI needs to deal with multiple structures, and proposes many solutions for overlapping markup (see section 1.3 for details), while IML describes the *basic* structures of a text (see section 1 for a deeper discussion about the content/structure relationship).
- *patterns adoption*: IML strictly follows few patterns, while TEI does not. I am not saying that TEI is a bad-engineered or pattern-unaware language. Rather, that my patterns are not enough for what TEI needs to express. As I said, it is a descriptive/prescriptive language aiming at capturing and forcing details, while IML is a descriptive/general solution.

The point is that TEI and IML have two different objectives, and were consequently designed on different principles. Renear et al.[RDM96] explained that 'the a text as seen by the SGML community is not the same as the text seen by the TEI community – that is, the accounts that they would offer of a text's structure are significantly different'. Although he mentioned SGML, similar considerations can be extended to IML, which indeed is nothing else that a (pattern-based) subset of HTML.

3 ISA*: A flexible architecture based on Pentaformat and IML

The Pentaformat model and IML can be combined to build a simple and flexible architecture, ISA*, which generalizes the ideas developed in the ISA project[Vit03]. ISA is a web application developed at the University of Bologna and designed to simplify and speed up the creation of web sites. ISA* has been applied to a variety of scenarios like web editing, collaboration, e-learning and professional book printing. In the development of the system, IML, and the pattern based approach described in this thesis have played a central role. Now on I will then use the first person plural to indicate the research group I belong to.

ISA allows users to easily produce and publish web content. Authors write content in MS Word (and specify the role of each text block by styles) and the system automatically converts such content into graphically advanced pages, by exploiting associations between the layout area names and the content styles, previously created by a graphic designer. ISA transforms such information into an XSLT that, in turn, will apply the selected formatting to the original Word content. ISA* applies a similar approach to heterogeneous domains and data formats.

This architecture (shown in fig. 1) separates all components of a document, then works separately on each of them, then recombines them again for the final output. The whole system consists of bi-directional converters from and to any existing data format we want to support. Our basic idea is that any application of digital publishing can be chiefly considered as a smart conversion from a source format to a destination format, with a little bit of application logic in between.

Diaz et al.[DWB02] proposed different models to perform conversions between data formats: *direct model* based on bidirectional transformations from a data format to another one, *intermediate format* model based on a new generic format used as an intermediate representation of any format to be converted and finally *ring model* in which data formats are virtually ordered in a circular structure and the transformation happens jumping from a format to the following one toward a pre-defined direction. The second model, a.k.a. *superior standard model*, proved to have a number of benefits in terms of efficiency, quality and implementation facilities, as confirmed by Milo and Zohar[MZ98] (whose intermediate model exploits similarities between documents' schemas) and Abiteboul et al.[ACM02] (whose intermediate model integrates heterogenous databased applications).

IML is therefore used as the intermediate data format that captures only relevant information of the input documents and ensures high-quality output by delegating the rendering to external tools. Currently, the formats we manage include HTML, MS Word, MS PowerPoint, InDesign, Open Document Format, DocBook, PDF, LaTeX, plain text, wiki-oriented formats, as well as arbitrary XML. All document workflows using this approach follow the same general steps as shown in the picture: *content extraction* (on the left) and high-quality *post-production*.

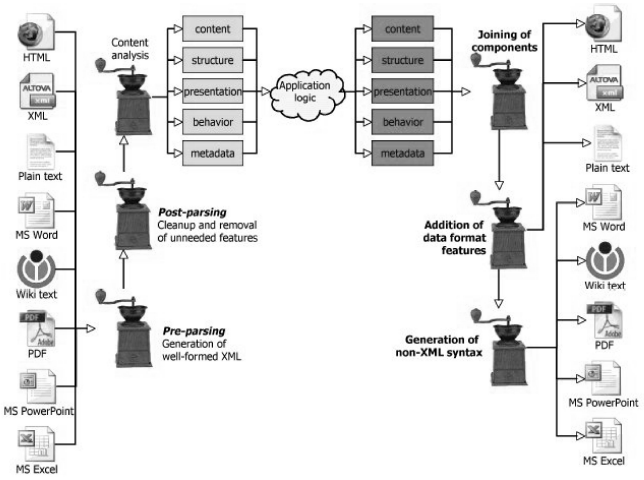

Figure 1. The ISA* architecture

3.1 Content extraction

The first step consists of extracting all the constituents of a document and normalizing content in IML, in order to be completely independent from the input format. That segmentation process can be further divided in *pre-parsing*, *post-parsing*, and *content-analysis*.

3.1.1 Pre-parsing

Since our architecture is intrinsically based on XML, our fundamental tools are converters from one XML format to another. While some of these converters work already on XML, others (such as MS Word or PDF, for instance), require a further step. Then, the first possible action we take is making conversion into XML, before any further content accommodation takes place. This operation, called *pre-parsing* (as it happens before we can actually have a chance to parse the resulting document as XML), is heavily dependent on the actual syntax being employed by the data format being converted and it is heavily different among data formats.

For instance, in PDF we first convert the binary PDF into exactly looking SVG, and then proceed to parse the document. With MS Word, we either fix the well-formedness of the HTML that MS Word automatically creates, or use directly (when possible) the WordML XML format. Further approaches (such as reading the RTF or the .doc format, or firing up the Word application to save the document in the required format) are being considered.

3.1.2 Post-parsing

After the pre-parsing step we read the resulting XML file and clean up the content and remove parts that surely will not be needed for the smart component separation. These operations, cumulatively known as *post-parsing*, are also different across data formats, but for semantic rather than syntactical reasons. These include re-joining lines into paragraphs for PDF files, or removing alternative variants of the same image in MS Word documents. Although these operations are still dependent on the quirks and peculiarities of each data format, they are done on an XML source, and therefore they can be and are usually done via a transformation XSLT style-sheet.

3.1.3 Content analysis

Pre-analyzed content is then scanned to identify individual features and denominate the constituents. The output of this phase is the same XML document that was provided in input, with additional attributes specifying whether an element is content, presentation, behavior or metadata. That is the central part of the input system, which actually separates the constituents of a

document, and allows downstream applications to selects only those constituents that they need to deal with, and ignore the rest. The current engine is completely generic and parametric, and it is based on XSLT technologies. Additional features in the conversion, or additional source formats to convert from, require only an update to the set of XSLT stylesheets. The output is a document where the generic parts can be easily converted into an IML document, and where everything that is not part of the IML file is either presentation or behavior.

The key point here is that any document can be passed to the engine, without imposing constraints on its internal structures and styles. The approach follows a "Garbage In, Garbage Out" paradigm: Isa*can extract content and reflow it from any source into any layout and no input file is rejected, but the better structured is the source, the finer will be the final output.

Different possibilities exist to ease this information extraction. One extreme is to impose strict rules onto the authors, possibly enforced with macros that verify if they are following them. In this case, editing is not free of hassle, but the conversion is perfect, simple and straightforward. The opposite extreme is to have the system accept just any document and do its best to extract the actual content; in this case, the complete freedom in writing has heavy impact on the sophistication/complexity of the converted result. An intermediate solution, that we have tested in the e-learning context described in [DIFM+05][DIFM+06], consists in giving the users a set of guidelines about how to use styles and input macro, and then in implementing the appropriate transformations with flexibility. Therefore, all documents can be processed by the system but, the more compliant they are to the guidelines, the better will be the final result and the correct reformatting.

Whenever no guarantee of correct input is available, ISA* applies a set of heuristics and analysis techniques that allows users to provide even unformatted documents to produce well-formatted output[DIVCV04]. These heuristics, expressed as conversion rules in an XSLT metastylesheet, can be adapted and polished so as to make this approach both flexible for different scenarios, and powerful for different levels of output complexity.

3.2 High-quality post-production

In the second part of our architecture the perspective changes radically: what is an abstract description of content, have to become an actual file, in a specific format, with specific formatting and layout. That process involves two clearly distinguished sub-processes: *application logic* and *high-quality rendering*.

3.2.1 Application logic

Once separated, the specific application can act on the constituents independently. Operations can vary considerably, from simple to more complex ones, depending on the purpose of the application itself:

- Simply repackaging it in a different format in order to convert a document from one format to another
- Substituting the presentation constituent with a new one in order to reformat a document, regardless of its source, with a completely different layout and final aspect
- Substituting the content constituent with a new one in order to reuse the presentation to create a new document looking similarly to the original one.
- Analyzing the structure constituent looking for specific types of content in order to filter it out (e.g., removing advertisement from a web page)
- Adding specific ontology-driven elements to the metadata constituents so that the document can be correctly placed within a workflow process regardless of its source format.

The list can obviously go on and on. What is important is the fact that all those operations are independent either from the input format or from the final one, since the files we work on are IML fragments, or abstract data about behavior, presentation and metadata.

3.2.2 High-quality rendering

Finally, all ISA* tools take care of re-generating a final document ready to be delivered to the final application. These processes follow a sequence absolutely symmetrical to the initial one: the new IML document is enriched with data format-specific information, and then converted via XSLT stylesheets into an XML format which can either be the final format, , or the input to a converter to some kind of binary format, assuming we have the correct converter from XML to binary.

Particularly important in this stage is the quality of the final conversion. The final rendering step takes in input both the converted document and configuration parameters that express the quality requirements to be met. By applying adaptive models, the renderer transforms the IML content into a ready-to-publish output, for instance a reusable and accessible learning object based on SCORM, or a sophisticated XSL-FO file.

Our model suggests then using external and format-dependent applications that are smart enough to take sophisticated decisions in order to generate high-quality results. The complexity inherent in such high-quality results depends also on the sophistication of the formatter that actually produces the final artifact. The more powerful and reliable is the renderer, the lesser is the effort required to produce high-quality products. Theoretically, the author can be left completely unaware of all the technical difficulties, allowed to directly generate high-quality material with one click.

However, in many cases the results that can be obtained with existing tools are not sufficiently sophisticated for professional use. For this reason, we often need to improve or re-implement renderers, in order to obtain the required sophistication.

Chapter 6

An open publishing system: IsaWiki

The ISA* architecture introduced in the previous chapter is completely independent from specific data formats and from specific domains. In order to assess such architecture, as well as the wide applicability of the Pentaformat and IML, we have implemented different applications based on that model. The applications currently work on e-learning, professional printing, web editing and collaboration. More than the actual domains we selected, the key aspect is that a pattern-based segmentation model makes possible the implementation of advanced conversion tools, able to empower and simplify authoring processes. The first field of application was the World Wide Web. In this chapter I will present IsaWiki, a system which exploits the flexibility and multi-format facilities of ISA* and IML in order to 're-open the web-authoring' case. IsaWiki is a very complex system that could be a dissertation in itself. A detailed discussion is out of the scope of this thesis but a brief description is useful to explain our vision and application of a multi-format publishing model.

1 Re-opening the 'web authoring' case

IsaWiki implements a new model for web authoring that not only helps bridging the overlap between the reading and writing processes of web documents, but takes a step towards the full integration of reading and collaborating too. The proposed model of editing passes primarily through two different and complementary steps: on the one hand, the simplification of the publishing process in order to ease the creation of web documents, that we have called *writable web*; on the other hand, the improvement of the collaboration towards a new environment where all web users can easy collaborate on all web resources, that we have called *global editability* (or *web-wide collaboration*).

1.1 Writable Web

The publishing model of the World Wide Web is still asymmetric: apart from some exceptions each role (reader, writer, graphic designer) is different and independent from the other ones, it requires different skills/tools and it acts within a different step of the authoring process: the writer prepares in advance content, the designer prepares in advance layouts and finally an automatic (or manual) process merges them into the final document to be published. Afterward the reader can access (but not modify) the published content. Clearly this complexity makes publishing content on the web a voluntary and laborious act, and not one of the daily actions or a side effect of our daily intellectual life. In few words: most of our tools allow us to write *for* the web, not *on* the web.

Something is changing with the last trend of the World Wide Web. Recent evolutions allow users to publish for themselves many kinds of data. Users can create their own pages by exploiting WYSIWYG editors (like FCKeditor[Kna03]), or wholly client-side publishing environments

based on Ajax technologies (like LesserWiki[Yat06]), and so on. The distinction between producers and consumers is finally starting to be erased. Moving towards a really *writable web* just means following that direction, and allowing any user to write and manage web content with the same facilities used to read them.

The first issue to be addressed is the selection of the user interface. Two approaches can be considered: the *browser-based* and the *word-processor-based* authoring paradigm. In the first case, the editing process happens completely within the browser: the whole page is displayed in the browser window, so that the user can modify the actual content of the page or the whole document, though a WYSIWYG editor (see for instance Amaya [W3C01], or Ajax solutions, integrated in most applications, among which the same wikis). A WP-based approach is equally interesting and powerful, in particular when users want to re-use and publish legacy material. It consists of letting users to edit content with their preferred tools (for instance MS Word) and letting the system import and convert the original document into a however different data format and layout, through a powerful templating engine, as implemented by our application ISA[Vit03]. Other examples are systems (like JotSpot[KS04]) that include importers and allow authors to directly upload their content on the repository.

The second issue to be addressed, in fact, concerns the actual steps required to publish a web page. FTP connections and direct logins onto a web server are not simple enough: a user should be able to edit a page and publish it onto the web server directly from the browser or the editor, without separate tools, interfaces and processes. Wikis already adopt such solution but they are limited in the final graphical effects (and in the writing syntax), while weblogs are not flexible in the content creation. A possible solution could be WebDAV[GWF+99], an extension of HTTP that provides users methods to write and manage files directly on remote web servers; however it still requires expertise to be installed and configured. The best solution to this issue is just allowing users to edit pages within the browser and saving changes directly onto a web server.

One of the key aspects of such a writable system is a strong separation between content and presentation, in particular by providing support for a fine-grained storage of assets of content and for a systematic reliance on templating mechanisms for re-flowing any content in any layout. It is evident how the Pentaformat model fits directly such approach since any segmented constituent can be processed and re-flowed indipendently. Through such a system authors simply prepare content paying no attention to presentational aspects and using their preferred browser/editors, while graphic designers prepare layouts with a drawing application, possibly without any direct intervention in the underlying HTML coding. The system would then take the whole templating process upon itself. According to the time the templating engine acts, two different templating models can be identified:

- *Pre-templating*: the designer prepares in advance a layout, directly displayed in the browser window while the writer is creating content. Thus, the areas to be filled with content are activated and the writer looks at the final effect of the page during the whole authoring process. For instance, this approach is used by the WYSIWIG browser/editor mentioned before.

- *Post-templating*: the author prepares in advance content and saves them into an appropriate position on the server, and the graphic designer prepares in advance a layout. Before publishing the page onto a web server or directly whenever the page is requested, the system merges the two components into the final document. This approach is widely used by common content management systems (Boiko[Boi01] published a very interesting review about CMSs) or by tools supporting stand-alone content pages, like ISA[Vit03].

A point is very important: even if an unskilled user can easily create web content with such system, no limits exist for the productiveness and expressiveness of an expert one. The writable web model aims to decrease (up to the point of effacing it) the set of skills and knowledge required to write web content, but it would let skilled users produce the same advanced and structured pages which can be found today surfing the web.

1.2 Global Editability

Unlike web authoring, in whose field many interesting results have already been achieved and good groundwork exists today for a writable web, collaboration over the WWW is still quite difficult. The goal of IsaWiki is providing users an editing environment the current WWW to support a really web-wide collaboration, called *global editability*[DIV05a]. The *global editability* is a place where any user can share and collaborate on any content, regardless of locations, formats and access rights.

The need of customizing and collaborating over other peoples material as well as the belief that external and free modifications can improve this material is grounded in the very roots of the World Wide Web: the ancestor of the Web, Xanadu [Nel87] by Ted Nelson, was meant to be a global publishing environment where all users could access, read, re-use, modify and comment any material of any user, tailoring it to their own purposes. Many other systems developed by the Open Hypermedia community in the first 90s (when the web was in its infancy) allowed users to add external links, annotations and other complex structures on the top of any document, like MicroCosm [DHH+92] or Hyper-G [AKM95]. Some of them were extended to work on the Web, (for instance DLS[CDRHH95] or Webwise [GBS97]), new ones were developed in their wake, such as Arakne [Bou99], and the same Nelson proposed a way to integrate Xanadu with the (at the time) current web standards and languages, called "New Xanadu for the Web".

Nevertheless the World Wide Web has later gone on a different track ignoring some of the most relevant features of a really global and democratic collaboration platform[BVA+97]. Today the Web is an irreplaceable platform to read information, provide services, and connect remote people and resources but it still has not completely developed its potential as collaborative environment. Unexpected and unpredictable interactions can spontaneously result if any web user is allowed to edit and collaborate on any web page. Such approach can be likened to the open-source philosophy[Ray99], according to which several revisers and developers contribute towards the same task and everyone share skill and knowledge with the others. Open-source movement is increasingly gaining more importance, up to be considered by several researchers and professional as a new form of organizing knowledge work and making business. While the process of the production of software requires an engineered and methodical approach, a more dynamic and unpredictable collaboration can be equally useful and suitable in the case of multi-authored text documents. Similarly, the huge pool of users involved in the World Wide Web, can be transformed into an (almost) infinite pool of collaborators.

A partial step towards such an evolution is certainly the *external collaboration*, i.e. the possibility for any web user to add external comments, notes and links to any web page. In this case, the original document remains unmodified on the origin server, while the contributions are stored in external link- and data- bases and added to the document on request. Two standards were primarily proposed by the W3C to support such collaboration, XLink[DMD01] and Annotea[KKPS01], and a lot of systems exploit on them, like Goate[MA02], XLinkProxy[CFRV02] and XLinkZilla[DIMV05]. Yet, external annotations and links meet only some of the desired functionalities of a sharable editing environment, since users can only add a single extra layer of annotations onto the original documents, rather than freely collaborate on the document production.

A different form of collaboration, that we called *indirect collaboration*, can be obtained through the creation of personal anthologies: a web user surfs the web and freely collects data and references to the accessed content, in order to merge them into personal anthologies. The benefits are clear: the whole web becomes a global knowledge base from which everyone can draw content and ideas and any web author is an implicit collaborator of every other one. The qualifier indirect just means the fact that the collaborators do not necessarily know their materials have been included in other documents. Thus everyone can determine one's own navigation path, collect real anthologies of existing content or create new pages composed by personal contributions and other peoples' material as foreseen by Vannevar Bush [Bus45] and, more recently, HunterGatherer [SZM+02] and SMR[MBCKH03]. Obviously such a content sharing and aggregation has to be realized in a *controlled and safe* way, that is keeping traces to the original content and authors:

for instance, if all the connections between the original materials and their copies in the anthologies are kept in the anthologies themselves, everyone can trace who is the original author of the imported fragment, where the original document is stored and what any document is composed of.

Both external and indirect collaborations allow a single document to be enriched with ideas and content belonging to any web user: different knowledge, different opinions and ideas, different cultures and educations come together into the same resource. However, such external contributions are of course strongly distinct from the original content, but they are simply added or imported as an extra layer: there is no actual intervention on the content of the page, no deletion, no change in the overall structure and so on. Yet, within an open collaborative system all users should be allowed to freely customize and reuse content without limits of size, authorship and access rights. Customizations and collaboration, in fact, cannot be limited to an extra layer of data integrated into the original page, but they have to be considered as complex and unpredictable interventions that can occur without particular constraints.

In this sense, the final goal is the *global editability*, the creation of shared customized versions of web documents, where different contributions from different sources and different authors come together, regardless of write rights, skills and data formats. A simple scenario of this idea shows a reader that, while normally surfing the web, finds a document needing some comment or change and customizes the page (even better if the editing is performed directly within the browser), creating a new version of the resource. If the user is one of the authors of the page, a new official version is created, which is immediately made available to any other web surfer; otherwise a new personal variant of the resource is stored on a different server and is not made part of the official tree of versions. Any web user can, in turn, customize the new variant and create additional different personal variants, and so on. Finally, according to the decisions of all the authors involved, these personal contributions can be merged with the original content, thereby becoming official, public versions of the documents.

The design and implementation of global editability is a very complex task since tricky technical and non-technical issues exist and need to be deeply investigated, issues related to scalability, versioning, coexistence with the current WWW, management of digital rights and so on. They are out of the scope of this thesis (details can be found in [DIV05a]). What is relevant here is the fact that a global editability model presupposes an advanced and fine-grained management of the documents' fragments: each document should be divisible into a set of assets storable, manageable and reusable in independent ways. Moreover such segmentation must be independent from the format and layout of that document. The Pentaformat model and IML can be then exploited to handle such heterogeneity.

We can use IML as intermediate generic data format with features from both source and destination formats, and we can implement conversion of each format to and from this one. Any document can be normalized into IML and any further process can be performed only on that normalized content. Indeed conversion must extract the structured content and all the details about the layout and the style, and other dimensions of the Pentaformat model.

2 Taking ideas to implementation: IsaWiki

IsaWiki[DIV05a][DIV04] is a publishing environment that allows any user to create content for the web, to edit them directly within the browser, to customize any web page. It is a client/server system where some client-side modules (installable into the common web browsers) and multi-service servers (installable onto any web server supporting PHP) take part in providing services for registered users. The system naturally coexists with the architecture, the protocols, the languages and the tools of the current Web. All the possible interactions among the IsaWiki modules and the web clients and servers are summarized below:

- An IsaWiki server, firstly, is a plain web server delivering contents to any web client; on the other hand, it provides advanced services for publishing, customization and collaboration

to the subscribed users only. The IsaWiki server consists of a set of PHP scripts (running on PHP $>$ 4.0) and some XSLTs to perform conversions.

- An IsaWiki client surfs and downloads web resources; in parallel it communicates with an IsaWiki server and allows user to create and edit web pages within the browser. The IsaWiki client is developed for Internet Explorer 6.0 under Windows and for Mozilla and Firefox 1.0.3 for Windows, Linux and Mac OsX, and recently for Safari on Mac OsX.

A user interested in IsaWiki would simply install the plug-in and register himself onto an IsaWiki server. From then on, all the editing and surfing activities would be supported by the system.

2.1 The role of IML

The whole IsaWiki system relies on the methodological and systematic segmentation of content, structure and presentation of the Pentaformat model, and the ISA* architecture: any conversion is expressed as a normalization into IML and then a transformation in the destination format. Remind that transformations of IML are not "literal translations" of the content, layout and graphics (as it happens in case of a printer driver) but a smart re-flow of the mere content of the document into a different format and layout. The role of IML within IsaWiki is summarized in figure 1.

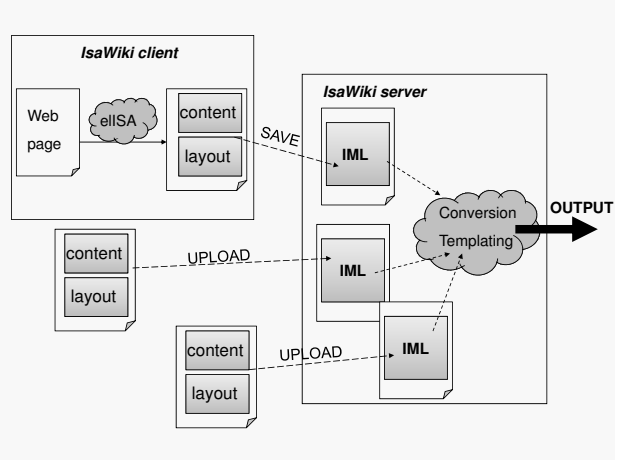

Figure 1. The IML-based conversion model of IsaWiki

IsaWiki implements both *browser-based* and *word-processor-based* authoring models (see section 1.1 for details). When a page is edited through the WYSIWYG editor the identification of the actual content areas (to activate editing facilities only on them) is performed by elISA[DIVCV04], a web pages' structural analysis tool, able to identify the real content of a page, by studying structures, patterns and regularities in the HTML code. It is actually implemented as a rule-based transformation engine that, once applied to HTML pages, adds appropriate attributes and other markup to the original document to indicate the role of each fragment. Actually elISA is able to gather the role of other page elements, but in IsaWiki it is simply used to determine the content areas, the only areas which can be modified. Thus, it is in charge of performing the *content-analysis* phase of ISA*, and extracting IML. When authors upload documents directly on the server, a server-side converter performs tasks similar to the elISA's analysis and extracts an IML representation from any document, if stored in one of the supported data formats. The client- and server-side analyses are temporary separated but we are working on a complete integration and sharing of extraction rules.

The *post-production* phase of ISA* depends on the output format, and it is implemented by different sub-converters, one for each supported format, which take in input IML. The most used in IsaWiki is the HTML engine, and it is directly imported from ISA [Vit03], the project which gave the name to the whole ISA* architecture. The same name of IsaWiki clearly indicates an integration between ISA and wikis.

2.2 Writable Web with IsaWiki

Figure 2. The IsaWiki editor for Safari

The first step towards a writable web is inevitably the provision of a simple and usable interface for surfing and editing. Figure 2 shows the IsaWiki client on Safari. Creating and publishing a new web page with that interface is simple and fast: by clicking on the relevant button, a list of available layouts appears and, once the user selects a template, an empty document is displayed in the main browser window. These documents are directly retrieved from the server on which the surfer is subscribed, by the sidebar that works behind the scenes. A set of predefined layouts is stored on any IsaWiki server after the first installation, but new ones can be easily added. Once returned to the browser, the empty document shows empty areas to be filled with content and bordered in red. Only these areas (content areas of the Pentaformat model) can be modified by the user, while the other areas remain visible in the browser window for contextualization and orientation. Buttons and forms to insert and update content elements are shown in a WYSIWYG editor integrated in the sidebar. Figure 3 shows an editing session with the IsaWiki editor for Internet Explorer. Note that the text editor has few buttons, corresponding to the few objects of IML. The menu to assign a role to paragraphs and in-lines, deserves attention since it corresponds to one of the core aspects of IML, the usage of the *@class* attribute.

Within the editable regions (that contain the normalized IML content) any content and structure can be freely modified, deleted or added by the user and the DOM is consequently and immediately updated according to these modifications. After the editing session and the input of some metadata the user has to simply click on a `save` button. Once again transparently, the sidebar post data to the IsaWiki server and the page is immediately published and available to the other web surfers. The editing facilities can involve subsequent changes too, since the system allows users to modify any subsequent version of the page, by using the same WYSIWYG editor. Any intermediate state of the document is recorded and any intermediate version can be requested and edited, so that the whole history of a document can be managed directly within the browser.

Any version of any document in any format can be accessed through a specific URL. For instance, by asking for `http://serveriw/index.doc_12.1` users can display versions 12.1 of

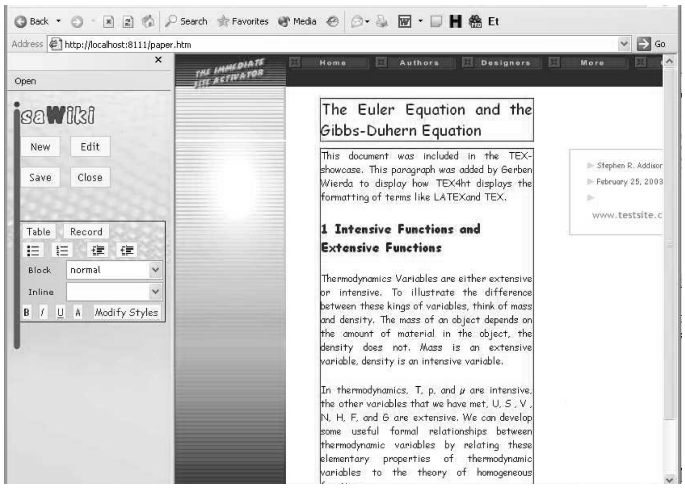

Figure 3. A WYSIWYG editing session on IsaWiki

`index.html` in `.doc` format, while the version 12.2 in XML can be requested by surfing the page `http://serveriw/index.xml_12.2`. Actually IsaWiki supports bi-directional conversions to and from MS Word, Wiki, HTML, TeX, XML and (partially) PDF files. All these conversions are performed through an intermediate IML segmentation and representation.

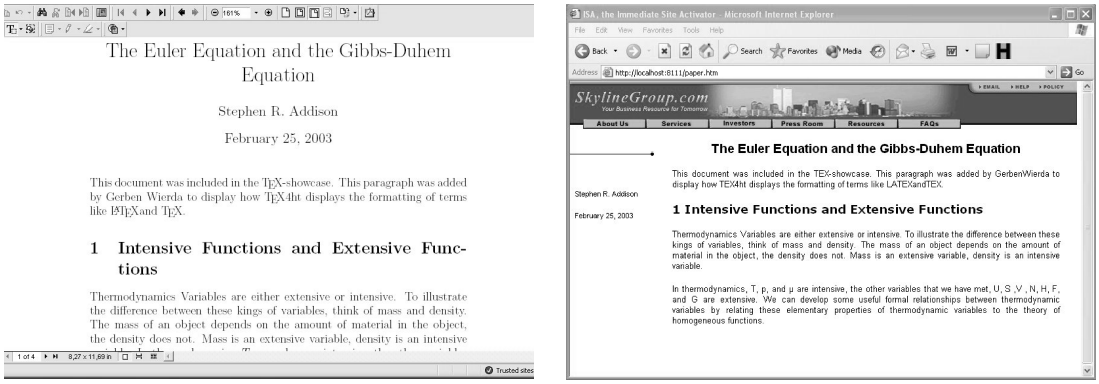

Figure 4. A PDF document re-flowed into an HTML page by IsaWiki

In figure 4, a PDF document is shown in its original form and converted into HTML via the intermediary IML reduction. The original document has been segmented according to the Pentaformat model, and the only (IML) content has been re-flowed in a different presentation.

HTML pages obviously play a relevant role, since IsaWiki is first of all a normal wiki, whose pages can be retrieved by any HTTP client. If properly configured, in fact, the IsaWiki server adds some links to any page useful to display a complete list of the documents on that server, a list of recent-changes, a list of the users and a search-engine. As expected, all these functionalities for the organization, searching and management of the local documents are independent from the format of documents (since they are performed directly on the filesystem or on the converted IML document).

IsaWiki users can write documents and upload them directly on the server via FTP or direct access (*WP-based* authoring model). An uploaded resource lives within the system as it had been created through the WYSIWYG editor and in particular it can be versioned and converted in any data formats. Whenever a resource is uploaded, since any intermediary revision contains

metadata about the revision number, format and authors, a server-side daemon positions the new revision within the server file-system. Many different scenarios can occur: a user, for instance, can create a MS Word document and manually save it on the server, after which any other entitled user can edit the same document in HTML (though the IsaWiki client) and, in turn, another one can re-edit the same page in .DOC or LaTeX, and so on. All in all, IsaWiki allows users to view and modify documents with their preferred tools, according to their preferences and needs.

To complete the multi-format vision, IsaWiki provides differencing and versioning engines, independent from the formats of documents. Regardless of the format a document was originally stored, two versions are first converted into IML and then the diff is taken. Finally a template is applied to the delta, so that a readable and clear document containing differences is returned to the browser. Fig. 5 shows a diff between two versions of the same document: although the output is an HTML page the first version had been uploaded as MS Word file, while the second had been modified through the WYSIWYG editor. Similarly, users can display differences between a wiki page and a LaTeX file, re-flowed into a simple HTML page.

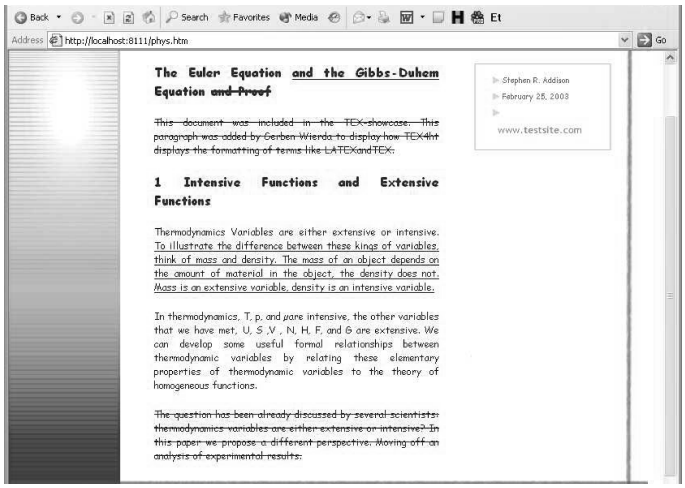

Figure 5. The multi-format *diff* of IsaWiki

2.3 Global Editability with IsaWiki

Besides being a platform for writable web, IsaWiki aims to be a collaborative editing environment based on a web-wide collaboration model. However, this side of development is still at its early stages, though the whole design is complete and claims to be strong and reliable: today IsaWiki implements a *restricted* and *distributed* collaboration model and provides *customization* of web pages. We are still developing the system to allow users to share such customized content and merge them into new multi-source and multi-authored documents.

IsaWiki is a distributed architecture based on HTTP, composed by common web servers (which coexist with the current web architecture) and client-side modules which communicate with these servers. A partial result is that the sub-web composed by these servers can be considered as a global environment for collaborative editing: the users registered on the IsaWiki servers can collaborate on the resources of these servers, directly using their browsers.

Whenever a user asks to edit the page in the browser window, in fact, the same page is shown with the same layout and presentation, but with red borders around the content areas (according to the Pentaformat segmentation). As expected, these regions contain the segmented IML content of that page and are identified by elISA, the structural analysis tool introduced in section 2.1; obviously, the elISA analysis is trivial on the local pages, since the content areas have already been identified by the templating mechanisms on the server, but it plays a key role with the

external web pages (as I discuss later, IsaWiki allows users to customize any web page, regardless of its origin-server, or better the content of any web page).

As for the creation of new pages, the WYSIWYG editor integrated in the sidebar allows user to wholly modify the content areas of a page directly within the browser and save new versions on the proper IsaWiki server. No direct access to the server file-system, no FTP connection and no technical skill is required, but the user has plainly to know how to use a WYSIWYG editor: whoever can use a browser and a simple word-processor, can collaboratively edit content. For any IsaWiki page the owner can define a set of users (or groups) entitled to create new official versions of the same page: not only the revisions made by the owner of the page but also any intervention of entitled collaborators will belong to the official history of the page.

If the author does not belong to the list of the official authors, IsaWiki supports *customization*. After any editing session, whatever the author is or not an official author, a new official version or a personal variant is created. The customization does not interfere with the normal navigation of uninterested users: when a registered IsaWiki user surfs a page, if a customized variant has been previously created, it is displayed, instead of the original page; that page remains unmodified on the origin server, available to the rest of web surfers.

Technically speaking, customization is possible thanks to the sidebar. For any accessed page, the sidebar catches the event and verifies (through a HTTP request to the IsaWiki server) if a personal variant of that exists. The communications between the sidebar and the IsaWiki server and between the browser and the origin server are performed in parallel and transparently to the user: this parallelism saves both time and user's patience with respect to a proxy architecture, as discussed in [DIMV05]. The content of the document from the origin-server is then modified according to the data received from the IsaWiki server (javascript and C++ functions are used to read and write the browser internal data in Internet Explorer, while Mozilla and Firefox use XUL[GHHW01] and javascript, and Safari exploits Object C). The end result is then displayed in the browser.

The most important aspect is that customization is not limited to the sub-web composed by the IsaWiki servers but can be extended to the whole WWW. Any page can be edited and any variant can be stored externally on an IsaWiki server, on condition that elISA can gather the content regions within the same page. All the documents that can be parsed by elISA (and so can be translated into an IML document), compose the documentary source of the IsaWiki collaborative editing system: in the future we plan to extend this documentary source to the whole WWW, but the core of the system will remain unchanged. Extending this source, in fact, requires only to strengthen elISA and to polish its capabilities in extracting IML. The Pentaformat model is powerful enough to segment (and extract content from) any document: what we need to do is implementing more powerful converters based on that model.

Once elISA has analyzed a page, whether it is a local or an external one, it can be edited through the WYSIWYG editor and saved. Server-side, IsaWiki implements a *session-based* versioning mechanism and external anchoring system. In particular, the content repository has a *customization area*, where each registered user has a personal space. In this space, for each personalized page, there exists a subdirectory (in case, within further subdirectories mirroring the original web site) which contains the whole tree of variants. Thus, any web page can be edited through the browser and any customized variant for any user can be added to the file system.

Figure 6 shows the home page of the European Journal of Information Systems and a customized version created with IsaWiki.

The key-aspect of customization is the ability to trace changes and distinguish personal interventions from original content. That is possible because customized variants are stored in external servers, so that any user can customize any page keeping credits of original authors and sources. The IsaWiki *diff* engine is able to calculate the differences between any version and the previous one, or better between their IML content, so that any contribution is clearly identifiable.

2.4 Still a long way to go

The implementation of IsaWiki is not yet complete. Customized variants, in fact, involve a single document (whose history can be fully and freely managed) but mechanisms to merge different

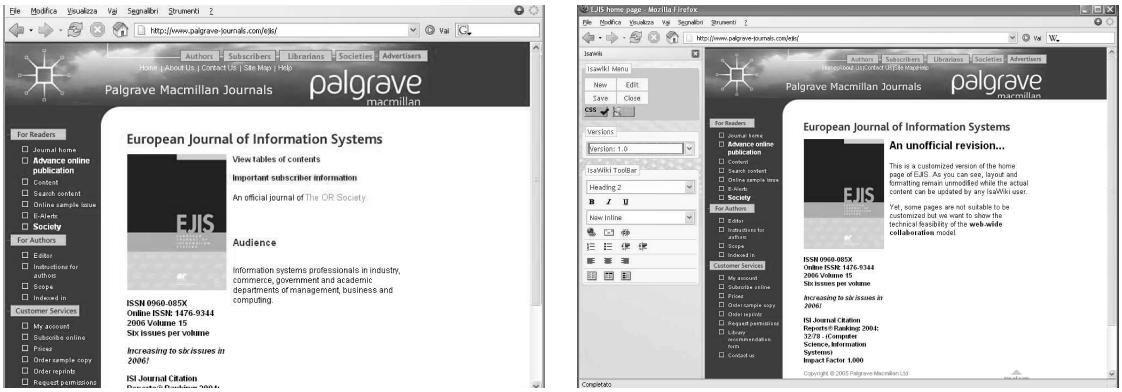

Figure 6. Customizing a web page with IsaWiki

resources into the same documents need to be still implemented, as well as many other functionalities which are currently supported only partially or still on our *to-do* list.

First of all we need to improve the main components of the system, in order to provide a more powerful support for the Pentaformat model. In particular, we plan to work on ElISA to really make *any* web page processable. ElISA can be seen as the 'entry point' for the system: the more that component is reliable and accurate, the more documents can be segmented (and the content can be extracted), the more the scope of editability can be extended over the WWW. In parallel, we plan to work on server-side analysis engines, both analyzers and formatters, in order to make automatic conversion more accurate and meaningful. The *diff* engine is another important point of development: our goal is computing differences between multiple versions and re-building documents where different contributions from different versions of different authors are displayed together. Similar solutions were already investigated in our previous project XanaWord[DIV03], a distributed editing environment which allows users to browse and edit pages by using common tools like MS Word and MS Internet Explorer.

Upon a solid implementation of the current features, we plan to implement advanced ones, to complete the vision discussed so far. A first step towards global editability is the implementation of mechanisms for verification, proposal and integration of private content into public ones: the user can freely create personal variants, later propose them to the official authors and, in case, integrate the personal interventions with the official ones.

The next step will be the support for a fine-grained management of assets of content. In the current implementation the atomic unit is a version or variant: versions are clearly distinguished, can be compared and converted from and to any format, but cannot be further segmented into reusable assets of content. Note that the segmentation into assets is different from the Pentaformat segmentation: it is a further step applied to one single dimension, the structured content, in order to identify and work on smaller pieces of atomic information.

We in fact plan to study IsaWiki as a global infrastructure based on IML fragments' management. Primarily we will integrate within the system a merging/import engine which allows authors to put together fragments from different sources into the same document, keeping information about each single fragment. This integration will allow users to create multi-sources documents (drawing content and ideas for all the accessed pages) or multi-authored documents (written in collaboration by many "emergent collaborators"). Finally, once this step is completed, a framework to handle intellectual property and copyright could be investigated as well.

In conclusion, there is still a long way to go and IsaWiki will probably stay with us for a long period. However, this early experience has already shown us how the authoring model of the World Wide Web can be actually improved by IML and ISA*, and how issues about web authoring, which seem to be relegated only to the ancient origins of the WWW, are still open today[DIV05b].

Chapter 7

Simplified authoring systems: IsaPress and IsaLearning

One of the most important advantages of the Pentaformat and ISA* is the strong separation between input sources and output ones. That distinction allows authors to produce very good output, without having to learn new technologies and tools. It is in fact the system that extracts the original content (regardless of its actual layout and formatting), and *automatically* convert it into a ready-to-publish and high-quality output. Such approach has been applied to professional printing and e-learning, in order to implement respectively IsaPress and IsaLearning, the two systems presented in this chapter.

1 ISA* for professional printing: IsaPress

Assuring uniformity and high-quality of their final products is not an easy and cost-effective task for publishing houses. Many manual interventions are still required to uniform source documents and make them ready to be ”digested” by a conversion process, being it completely automatic or hybrid. No matter if the final output is a book or a web-site, or a resource for a mobile device or anything else; what is desirable is that the process does not require, or at least that it minimizes, users' manual interventions and, above all, that the final results is still of high quality and can be directly published.

1.1 Issues in traditional professional publishing

The most widespread process to produce books cannot be still characterized as fully automatic, rather as a chain of semi-automatic transformations, that involves different actors with different skills: *authors*, who actually write a content (ignoring the final formatting, and having few technical skills about that), *publishers*, who decide the look&feel of the final product (coded into the publishing system by technical experts), *pagination experts*, who the content provided by authors into a format ready to be processed and printed (with professional tools like InDesign[Ado99]), and *typographers* who actually produce the final books. Note that I have omitted from the previous list roles like proofreaders, reviewers, editors and so on. The focus is not meant to be on the whole workflow of a publishing house, but rather on the semi-automatic process of converting and publishing documents.

A leading role is still played by the pagination experts who are actually in charge of importing raw content in the system and verifying that they can be really transformed into a well-formatted book. According to the complexity of the final output, as well as the number of constraints to be fulfilled, such experts have even to manual intervene on the content and, when need, fix errors. Fact is, software formatters are still limited and do not solve automatically the most complex issues, publishing houses require complex properties to be satisfied, content is very

often unforeseeable and full of exceptions, but without the (sometimes hard) work of pagination experts many books would not be published.

After the first pagination, the typesetter is the only person allowed to work on the book artifact, and becomes the pivot of the publishing process until the final delivery: copy editors, proof-checkers, indexers, etc. basically provide low tech outputs of their work (most often hand-scribbled paper versions of the first pagination of the book) to the typesetter who have to insert them **manually** in the typesetting applications.

1.2 A revised workflow with IsaPress

The abovementioned limitations of a traditional workflow can be addressed by implementing an automatic conversion system, that takes in input plain and unformatted text and automatically produces high-quality books. It is evident how such application is a direct customization of the ISA* architecture (see section 3 for details), where input files are produced by common word-processors, and output ones are high-quality PDFs or documents in other format for digital publishing, like DocBook. We called that application IsaPress. Fig. 1 shows the interface of the system:

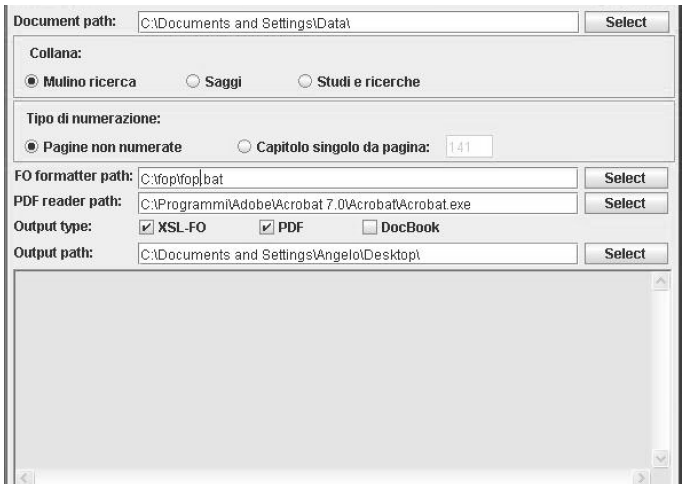

Figure 1. The interface of IsaPress

The current implementation is a stand-alone Java application but we are working to include it into a web application and to integrate it in legacy content management systems too. As expected, the core of the application is the internal conversion engine, while the interface is simply a help for users to invoke the process and can be easily changed or customized.

The whole process is performed automatically, thanks to the smartness and capabilities of the IML analyzer and the final formatter (which I briefly discuss later on) that actually produces high-quality material. Authors' effort is minimized, since they have to simply upload a MS Word file onto the system, and run the application. MS Word is undoubtedly the most widespread text editor, in particular among non-technical users, and several publishing houses use to receive MS Word files from their authors. By automating processes of conversion and tuning, publishing is simplified and sped up, and authors can import and re-use existing material with little effort. Yet, content verification and polishing could be required in order to make content ready for a book or a different output, but they are out of the scope of this work, being them purely editorial interventions.

Figure 2 shows an example of conversion performed by IsaPress: a MS Word file has been transformed into two very different PDF files, both ready to be printed.

The opposition between the simplicity of the input file, and the sophistication of the final ones

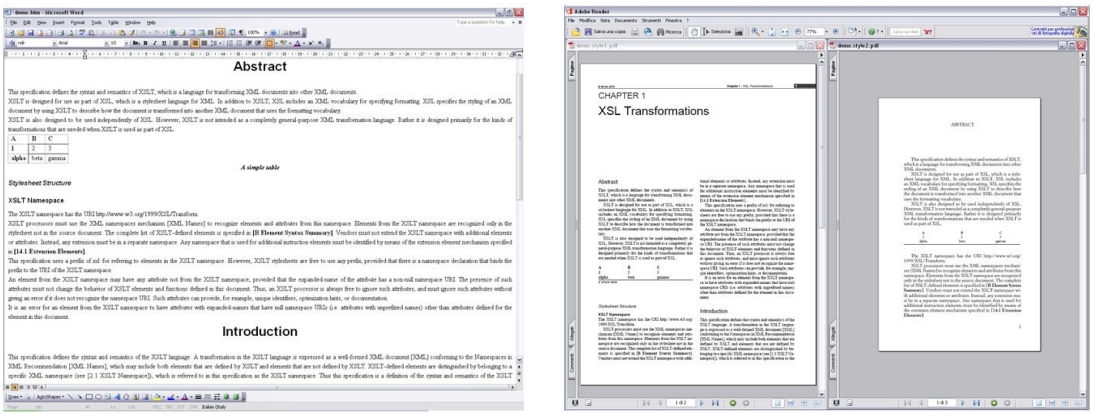

Figure 2. Two different PDFs from the same source Word document

is evident. One more point is worth being remarked, about the source file: it is basically composed by a sequence of classified paragraphs (though MS Word styles) and some basic objects like tables, lists and images, according to the IML model. IsaPress follows a GIGO ("Garbage In, Garbage Out") approach, described in 3.1: all documents can be processed by the system but, the more compliant they are to the guidelines (the more they use styles properly), the better will be the final result and the correct reformatting. The final results can be very different from each other, as shown in the example. IsaPress, in fact, takes in input a detailed description of the typographical features of the output, in terms of number of columns, dimensions, fonts, etc. Consider an example of a publishing house that produces series and essays: changing the typographical features of a series or the series where a specific content is published can be easily handled, by modifying those input data.

The main conversion of IsaPress goes from MS Word documents into PDF files. It has been the main focus of our research for a long period (and the ground where we obtained the best results), as well as one of the most useful applications. However, other formats are particular interesting in the field of professional digital publishing like DocBook[Wal99] or, among input formats, the XML exportable with InDesign[Ado99], a desktop publishing application widely used among professionals. We have generalized the original process of IsaPress in order to manage these formats too.

1.3 A magnifying glass on IsaPress

The internal architecture of IsaPress is modeled on the general ISA* framework. It is composed by independent and separately modifiable modules. Each module is characterized by a well-defined interface and communicates with the others by exchanging format-specific documents. As expected, two main phases can be identified:

- *Content extraction*: The document is processed by a java/XSLT engine which produces an IML file taking in input a MS Word or InDesign file. No particular plug-in is installed and no limitation is imposed over the features of the authoring tools.
- *High-quality post-production*: the intermediate IML document is then transformed into an XSL-FO file (including some layout extensions), according to an XSL-FO template given in input. At the end, the XSL-FO intermediate file is transformed into PDF, by exploiting a modified version of the FOP formatter. The conversion in DocBook is directly performed by a single XSLT.

1.3.1 Content extraction: from InDesign and MS Word to IML

The source parser is a java abstract interface, instantiated by specific parsers able to extract IML content from MS Word and InDesign. The following example shows the code of a (quite simple) input file for the MS Word parser.

```
<div class='Section1'>
 <div style='mso-element:para-border-div;background: silver;
 border:solid window-text 1.0pt; padding:1.0pt 0cm 1.0pt 4.0pt;
 mso-border-alt:solid windowtext .5pt;'>
  <h1 style='background:silver; border:none; padding:0cm;
   mso-border-alt:solid windowtext .5pt;
   mso-padding-alt:1.0pt 0cm 1.0pt 4.0pt'>
   <span lang=EN-GB style='mso-ansi-language:EN-GB'>Conferences.
    <o:p/>
   </span>
  </h1>
 </div>
 <p class=MsoNormal>
  <span lang=EN-GB style='font-family:"Century Gothic";
  mso-ansi-language:EN-GB'><o:p> </o:p></span>
 </p>
 <ul style='margin-top:0cm' type=disc>
  <li class=MsoNormal style='mso-list:l1
   level1 lfo3;tab-stops:list 36.0pt'>
   <span class=SpellE><b style='mso-bidi-font-weight:normal'>
   <span lang=EN-GB style='font-family:"Century Gothic";
   mso-ansi-language:EN-GB'>DocEng '06</span></b>
   </span><o:p></o:p>
  </li>
 </ul>
```

The file contains a title and a list of conference names (with a single item) formatted according to the authors' preferences. Actually it is a temporary `.htm` version of the original `.doc` file (where `.htm` is the extension of the HTML pages produced by MS Office applications), so that XSLT technologies can be exploited later on. The file is far from being raw content: apart from being bad-formed in XML, it is "polluted" by style information, MS Word-specific tags and attributes, irrelevant formatting data and so on. Different versions of MS Word, for different operating systems and languages, even produce very different source code for pages that look identical in the WYSIWYG interface. Moreover, the source could be much more complex in presence of crossing references, backtracking, drawings and so on.

The parser cleans up the document and performs the two steps depicted in section 3.1: (i) reduces the source file into a processable XML, and (ii) removes presentational aspects (besides behavior and metadata), in order to extract the actual content. The final IML result is the following:

```
<h1>Conferences</h1>
<ul>
 <li><p><b>DocEng '06</b></p></li>
 <li><p>WWW '06</p></li>
</ul>
```

Some filters perform a similar normalization exist, like WordCleaner[All04] or some plug-ins for MS Office, but we do not use them in order to let authors use MS Word, without any plug-in or external application. The main principle behind the system, in fact, is accepting any MS Word document without imposing rules to the authors. Moreover, they do not produce a completely

pattern-based fragment, as IML does, so that implementing further conversion would be more complex and error-prone.

The parser for InDesign performs a similar task. The last releases of InDesign allows users to export XML files. Since this format still keeps a lot of limitations on the internal representation of styles, tables, footnotes and boxes, we had to implement a pre-processor fixing those errors and making content digestible by IsaPress. That module could not be directly integrated in IsaPress since it has to modify InDesign internal structures at run-time. That is why we have implemented a plug-in, in charge of filtering and export content. Once filtered, content is passed to IsaPress which translates it into IML. The following example shows a XML fragment produced by the pre-processor, corresponding to the previous MS Word code. Note that it is not still normalized into IML and, for instance, lists are still represented as a sequence of classified paragraphs, and tag names follow a given (and not relevant here) prefix numbering.

```
<I_title>Conferences</I_title>
<II_unordered_list_item>
 <VI_bold>DocEng '06</VI_bold>
</II_unordered_list_item>
<II_unordered_list_item>WWW '06</II_unordered_list_item>
```

1.3.2 High-quality post-production: from IML to PDF and DocBook

In the ISA* architecture a relevant role is played by the final renderer, which is in charge of actually producing a high-quality output (post-production phase). This step is particularly complex with IsaPress, since the production of PDF files requires a further step of conversion from XSL-FO to PDF, performed by stand-alone formatters. Although the XSL-FO standard is (quite) complete, the support provided by existing formatters is not yet completely satisfactory. Then, we have implemented a customized version of FOP[Apa01], that meets high-quality requirements, called IsaFlex. Actually IsaFlex is a wrapper around our previous application[DIFV06]. IsaFlex is out of the scope my thesis but few words are useful to remark its irreplaceable role in IsaPress.

IsaFlex automatically paginates content ensuring high-quality, by exploiting users' suggestions and implementing customized *line-breaking* and *page-breaking* algorithms. The main idea is that the document itself contains additional information about the typographical characteristics that the application is allowed to modify in order to fill the pages without violating any keep constraint. We proposed in fact a customization of XSL-FO which allows users to express adjustments that will be later implemented by IsaFlex.

While generating a PDF file requires a two-phases process, the production of DocBook files is performed by a simple XSLT which (i) transforms plain structures into hierarchical ones, and (ii) maps IML objects into DocBook constructs. Those conversion rules can be configured, so that slightly differences in input and output can be handled with little effort.

1.4 Real-life use of IsaPress

IsaPress is not a prototype, but a working system used by an important Italian academic publishing house, called "Il Mulino", in order to officially publish books. The MS Word files produced by the authors are continuously revised, corrected and updated; after each change, an updated PDF version is automatically obtained and integrated in the overall workflow of the publishing house. They are actually used for reissues or publications in different book series.

Recently "Il Mulino" have been working on the digitalization of some legacy books and journals. DocBook resources are generated with IsaPress, and uploaded into a shared repository. Those files will be soon processed with our engines, and published as HTML and PDFs (of different series).

2 ISA* for e-learning: IsaLearning

E-learning is another scenario, which is increasingly gaining importance, and where the ISA* architecture has been applied. The problem is basically the same of professional publishing: high complexity in the creation of high-quality material, which still requires specific skills and tools to be generated. On the other hand, that complexity is quite different from pagination issues discussed so far.

2.1 Issues in producing high-quality learning objects

E-learning authoring is characterized by high technical complexity due to the parallel efforts of creating effective e-learning content and referring to all relevant standards. Such complexity is very often underestimated, and it is easy to mistake *e-learning material* and *high-quality e-learning material*. A set of slides on a web repository, a set of inter-linked HTML pages, even a well-organized pool of resources on a web server do not constitute high quality.

The simple creation of an LO, correctly accompanied by metadata (both embedded and in the manifest file, as required for instance by SCORM[Lea04]), already requires that many constraints are respected. If, furthermore, the content has to be available according to universality and usability guidelines, a larger set of rules apply. A higher level of complexity is required when accessibility must be considered (to meet for instance the requirements of the Web Accessibility Initiative, WAI[HB05], and the Web content accessibility, WCAG[CVJ99]). The harsh consequence is that really few authoring teams can produce by themselves learning objects that can meet *all* these requirements. Most times some such requirements will be neglected (to the detriment of final quality) or delegated to external experts before the final publications of the material.

The traditional approach to generate e-learning content is, in fact, based on a two-phase workflow: first, the author produces initial material in a source format (usually created with personal productivity tools) and then this collection of unrefined materials is processed with *ad-hoc* tools by a staff of experts. Due to their complexity, several activities have to be performed by the editorial staff.

Several difficulties can be identified in such a workflow, all ascribable to the same problem: the great *misalignment* between what the author edits and what is actually delivered on the final platform. First, tools do not usually support authors in the provision of all the required information in SCORM metadata, alternative descriptions or table summaries for XHTML compliance and accessibility support; someone else has to insert such data, possibly disconnected and desynchronized with the original authors. Second, the author could design courses that follows an educational model unsupported or only partially supported by the delivery platform; in that cases, unsolved features need to be amended by asking authors to change and adapt content to the limitations of the system. Third, and more important, content updates (as small-time typo corrections) need to be performed *directly* on the final LO, by exclusively using the authoring tools included in the platform; otherwise the editorial staff must repeat the conversion process again. Even little modifications require many steps to be performed and intermediate documents to be produced, so that the overall content maintenance is very difficult, error-prone and time-consuming.

2.2 A revised workflow with IsaLearning

The limitations of the traditional approach can be rather addressed by implementing an automatic conversion system, that takes in input plain and unformatted text and automatically produces high-quality learning objects. It is evident how such application is a direct customization of the ISA* architecture (see section 3 for details), where input files are produced by common word-processors, and output ones are high-quality SCORM packages, containing web pages compliant to standards and guidelines. We implemented such application and called it IsaLearning. Figure 3 shows the interface(s) of the system.

IsaLearning is the first module of a more complex system, called ISA-BeL[DIFM+05][DIFM+06], that completely automates the production of usable, universal and accessible e-learning material. IsaLearning (whose interface is shown on the left) is in charge of extracting IML from input files,

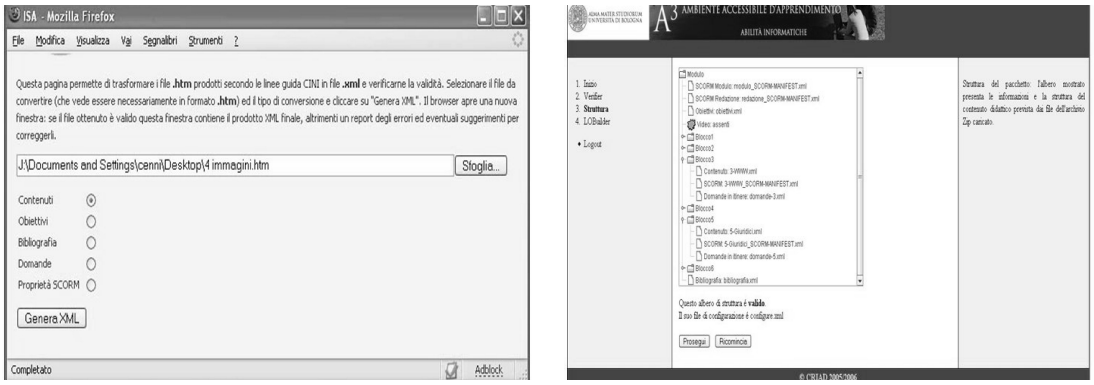

Figure 3. The interface of IsaLearning

and packaging intermediate XML documents for delivery by a module called WebLob. As expected, the core of the application is the conversion engine, while the interface is only meant to simplify users' interaction and can be easily customized or changed. The system currently in use was implemented in PHP, but a completely equivalent version in Java has already been coded and tested.

IsaLearning takes in input a MS Word file. The author simply has to create content, indicate the role of each content fragment (using predefined styles) and supply additional information such as alternative descriptions, acronyms, glossary terms, etc. Every authors action is performed through MS Word. The source file is then passed to the web application, that produces a specific XML file, through an intermediate IML representation.

Besides the format taken in input and the use of IML, there are many contact points between IsaLearning and IsaPress. First of all, the GIGO ("Garbage In, Garbage Out") approach (see sections 3.1 and 1.2): the more the input is well-structured and the text fragments are correctly marked-up, the more XML file is meaningful and easy to be transformed into a good output. However any file can be normalized into XML content, cleaned from presentational aspects. The second point is about the impact over the traditional workflow. Since the conversion is automatic (apart from some operations done by an editorial staff, once and for all) all changes on the sources are directly mirrored in the final output. Authors and editors do not need to learn new technologies and tools, but they only need to modify the source files over and over time, up to produce valid ones. High-quality and sophistication will be obtained by the following process, whose technical details they are not aware of. Thus odd jobs such as proof reading, last minute changes, major updates, special purpose customizations and similar small and big changes of content can be performed on the original documents and transferred into the output, without requiring complex passages among editorial staff.

Figure 4 shows a page created with MS Word and the same content how it appears uploaded on the ATutor e-learning platform[Gay01]. It is evident how the simplicity and minimality of the source document counters with the sophistication, uniformity and usability of the final output.

IsaLearning users receive an author-kit which contains a toolbar to assign a style (a role) to any text fragment, a toolbar to validate content (validation can be further performed with the web interface shown before) and one to insert metadata. Note that toolbars are not mandatory for making the system function, but they are only meant to be an help for authors. What is important is only that the input file correctly expresses fragments' styles and document's properties.

2.3 A magnifying glass on IsaLearning

As discussed so far, the internal architecture of IsaLearning is modeled on the general ISA* framework and it is quite similar to IsaPress. It is then composed by independent and separately modifiable modules, characterized by well-defined interfaces and format-specific exchanging docu-

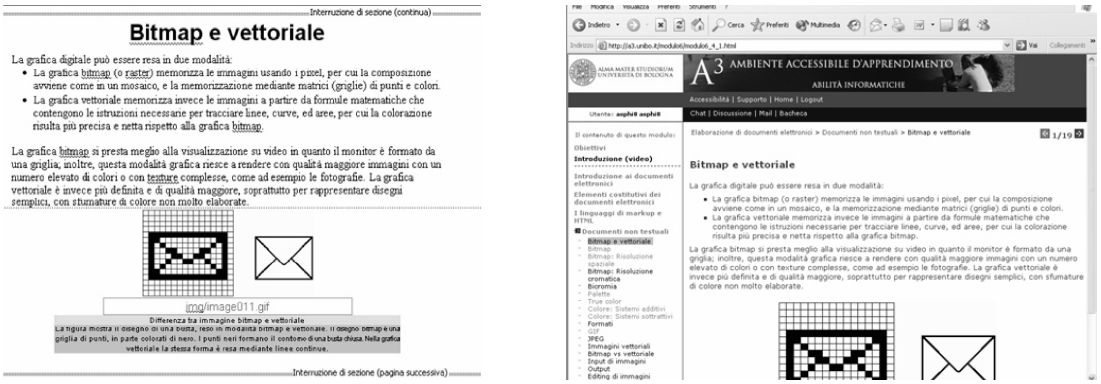

Figure 4. A MS Word file converted into an accessible LO

ments. Two main phases can be then identified:

- *Content extraction*: The document is processed by a PHP engine which produces an IML file taking in input a MS Word. The author-kit helps authors in meeting high-quality requirements.
- *High-quality post-production*: any intermediate IML document is transformed into an HTML page, and all pages are collected into a learning object, by the WebLOB module.

2.3.1 Content extraction: from MS Word to IML

The source parser is a PHP engine able to extract IML content from input files. As expected we have transferred the same heuristics and methods of IsaPress to IsaLearning, since they both process MS Word files. What changes is the PHP wrapper, but the actual logic remains the same, being it implemented via XSLT transformations (see section 1.3.1 for details).

Actually the fact that we provided authors alternative methods to express extra information (like acronyms, descriptions, summaries, etc.) had a strong impact on internal conversion, since the XSLT is in charge of rebuilding structures and data by collecting them from the original source. The following example shows a fragment of MS Word (once again I show a `.htm` file, obtained saving a `.doc` file as HTML page, to later exploit XSLT technologies), used to indicate the URL of an image to be included in the final LO, and a short and long description:

```
<div style='mso-element:para-border-div;
border:solid lime 1.0pt;mso-border-alt:
solid lime .5pt;padding:3.0pt 3.0pt 3.0pt 3.0pt;
margin-left:155.95pt;margin-right:155.95pt'>
<p class=imgURL style='margin-top:6.0pt;
margin-right:0cm;margin-bottom:0cm;
margin-left:0cm;margin-bottom:.0001pt'>Image.jpg</p>
</div>

<p class=descbreve>Relazione di precedenza o diagramma
reticolare.</p>

<p class=desclunga>La figura mostra un grafo orientato
con otto nodi numerati che rappresentano le attivit
per sviluppare tre moduli; la prima attivit, di
<b style='mso-bidi-font-weight:normal'>
<span style='color:red'>preanalisi</span></b>,
precede tre coppie che possono essere svolte in parallelo;
```

```
segue una attivit di <span style='color:red'>
<b style='mso-bidi-font-weight:normal'>test</b></span>.</p>
```

Presentational aspects are removed by the parser, as well MS Word-specific features. Moreover, from the sequence of classified paragraphs (note that the role of each of them is expressed by the `class` attribute) the parser extracts information about the image. The input fragment is then transformed in:

```
<img src='Image.jpg' alt='Relazione di precedenza o
diagramma reticolare.'>La figura...
attivit, di <b>preanalisi</b>, precede tre coppie
... attivit di <b>test</b>.<img>
```

Similar extractions are performed for acronyms (whose expansion is expressed by using footnotes), for table summaries (written in the first paragraph after a table), for glossary terms (explained in footnotes as well), and so on. Note that these features depend on the high-quality and complex requirements of the output: for instance, if users were not interested in accessibility we could omit a lot of complexity and analysis.

2.3.2 High-quality post-production: from IML to SCORM learning objects

The post-production process is performed by WebLOB. I do not want to add details about WebLOB (which is actually produced by another research group) but few words are useful to give a global vision of the system. It is a Java stand-alone application that performs a two-phases process of *composition* and *templating*. First of all XML files are transformed into XHTML valid pages and all the external resources are collected and put together, internal and cross-references are resolved and complex data structures (glossaries or exercises) are built. They all are packaged into a learning object. It also integrates into the LO multimedia recorded accessible video lectures, if required. The final operation is the configuration and application of templates previously created by professional designers. WebLob connects and merges them with the original IML content.

2.4 Real-life use of IsaLearning

IsaLearning is not a prototype: it is a production system, used for more than two years within and outside the University of Bologna. The initial sponsor and originator of the tool has been the A3 Project (*Accessible Learning Environment, "Ambiente Accessibile dApprendimento" in Italian, http://a3.unibo.it/*) which was carried out at Department of Computer Science of the University of Bologna to generate Learning Objects for the teaching of basic Information Technology skills. The original provision of 20 courses on IT, which are still being delivered to more than 2500 UniBo students every year, is now accompanied by several dozens of courses on all subjects, from Business to Biology, from Foreign Languages to Psychology, and a total of more than 350 LOs have been generated both for the University of Bologna and for a major Italian inter-University association. More than 70 content authors have received and have used the author kit.

The experiences of IsaLearning and IsaPress showed us the flexibility and power of a radically simplified approach in designing markup languages. ISA* and IML techniques have allowed the creation of two industrial strength production systems targeting Web course material and high-quality commercial printing (as well as a research system allowing collaborative web editing and modification, IsaWiki). Even if based on a very simple data model (IML) these tools proved to be powerful and useful, and the same patterns proved to be versatile enough to express (and capture *a posteriori*) high-quality content.

Chapter 8

Interoperability and interchangeability among wikis

The applicability of the segmentation model proposed in this work is not limited to the ISA* architecture, but holds whenever authors/designers need to translate content between heterogeneous data sources. Wikis are a very interesting field of application, due to their plainness: they in fact rely on a strong distinction between content and presentation, and on a limited set of constructs available to users. In this chapter I will discuss a solution to adopt patterns in the context of wikis, introducing the WikiFactory project, and a mark-up language called WIF. WIF is a language able to describe any wiki content, and interconvertible with IML, based on the same pattern analysis.

1 WikiFactory: from ontological descriptions to (semantic) wikis

Wikis[CL01] are collaborative tools used for fast and easy writing and sharing of content on the Web. They provide a simple, quick, informal way to create web sites, web applications, shared environment for discussion and document collections, tools for distributed cooperative writing, and so on. The strength of wikis is their free notion of web editing, empowered by some careful technical choices (versioning, direct editing in the browser, minimality); but they are still limited to maintain large web-site with domain-specific and recurring structures and data. Spontaneous contributions and open editing still need to be fully integrated with aided generation and management of content.

The creation of structured and domain-oriented publishing environments can be supported by the emerging Semantic Web technologies. In particular, semantic wikis have increasingly been gaining importance as means to integrate benefits of both free editing and semantics annotation. A semantic wiki is a wiki enhanced in order to encode more knowledge than just structured text and hyperlinks, and to make that knowledge readable by machines too. Several examples of semantic wikis can be cited: Rhizome[Sou05] allows users to create content with explicit semantics, with little effort, SemperWiki[Ore05] aims at integrating searching and indexing functionalities with a personal space for each user, SweetWiki[BG06] integrates a WYSIWYG editor, extended to support semantic annotation and as a fine-grained navigation interface. Volkel et al.[VKV+06] proposed an implementation to make content of Wikipedia understandable and processable by machines, by intorducing typed links among pages.

We figured out a different point of contact between wikis and semantics: using semantic information in order to generate wikis, apart from annotating them. WikiFactory[DIPV06b] is a framework designed for the automatic generation of wikis from ontological descriptions. The application takes in input an OWL description of the domain where the wiki will be used, written by different users with different skills, and translates such description into actual wiki pages.

Moreover it adds a plug-in to the wiki, in charge of monitoring users' activities and keeping consistency between the ontological description within the engine and the wiki instance. The first prototype of WikiFactory[DIPV06a] was a Java application aiming at demonstrating the feasibility and the potential of the WikiFactory's model. It worked on MediaWiki[Man02] and generated pages for that specific clone only. Then, we have implemented a more complex and stable version (that will be soon released under GPL), once again in Java, able to produce content for different wiki clones and to provide APIs for the actual monitoring of users activities.

What is relevant for my thesis is a particular feature of WikiFactory (actually I have also investigated ontology consistency and constrained editing models[DIZ06], not relevant here): the *independence from the delivery platform*. WikiFactory, in fact, aims at delivering the same content on different wiki-clones, by exploiting a cross-wiki segmentation of wiki pages.

2 A pattern-based segmentation of wiki content

The segmentation model discussed in chapter 3 is valid for wikis as well, since they are common HTML pages, when displayed in a browser. The same plainness of wikis and their editing (and templating) paradigms fit *directly* in the model I discussed. Two features are relevant:

- *strong separation between content and presentation*: when editing a wiki page, the only content of that page is shown in the textarea (or the WYSIWYG editor is activated only on the actual content), and authors work only that segmented component. Presentation is added later, through server-side templates.
- *syntax minimality*: few basic objects can be used when editing a wiki page. Moreover, although each wiki clone proposes its own syntax, a common set of objects can be easily identified, like paragraphs, lists, tables, in-lines and so on.

It is quite natural to think about a common language able to express whatever (any content, without presentation) can be written in a wiki markup language. In the literature some efforts toward interoperability among wikis can be found. The most important is WikiCreole[SSJC06], a common wiki markup language currently under development to be used across different Wikis. It's not replacing existing markup but instead enabling wiki users to transfer content seamlessly across wikis, and for novice users to contribute more easily. WikiCreole is meant to come alongside existing wiki syntaxes, and was designed to avoid conflicts. In particular, it uses a text syntax which spaces, asterisks, underscores symbols to indicate the role of each fragment. It is not based on XML, and it intentionally covers a subset of shared features among the most used wiki clones.

WikiFactory needs a more complex language, able to also express semantic relations among elements (mapping what the input ontology says). We decided to adopt WIF (Wiki Interchange Format)[DIV06] and I became a co-author of the language, with Max Voekel. The idea is developing WikiFactory and WIF simultaneously, in order to share issues and results between these two research efforts.

2.1 WIF: Wiki Interchange Format

WIF existed before WikiFactory, and was proposed for different purposes By Oren and Volkel[OV06]. The main motivation behind the language was finding a way to automatically migrate content from one wiki to another, and re-use existing material. In particular they focused on interoperability among semantic wikis. Four principles supported the design of WIF: (i) *extensibility* (the initial set of tags should be easily extended with new ones, for specific needs and requirements, like semantic features), (ii) *easy transformation* (a WIF document should be easily translated into existing wiki syntaxes), (iii) *easy creation* (a WIF document should be easily extracted from existing wiki pages), (iv) *easy rendering* (a WIF document should be easily and properly displayed in a browser). Then, authors proposed a meaningful subset of XHTML as core of WIF, and a predefined set of rules to package multiple pages into zip files, loadable onto any WIF-compliant wiki.

The innovation of WIF does not rely on new tags and names, rather on expressing the shared features and objects of m(any) wiki clones.

The initial description of the language was quite informal, since authors wanted only to foster discussion among the community. We then produced a first DTD of WIF and set up a wiki for that language[DIV06]. WIF 1.0 integrates previous experiences on semantic wikis and WIF, with the pattern-based model proposed in this thesis. Basically, it is pattern-based revision of the initial proposal, where each element follows one of the patterns discussed in chapter 4. Moreover, the set of elements and attributes included in WIF was influenced by the IML analysis. The following table shows WIF 1.0. Once again, I have used a tabular visualization to highlight the strict relation between patterns and WIF. It is not difficult to picture a more familiar DTD or Schema.

Pattern	Elements	Content Model
Markers	`img`, `hr`	EMPTY
Atoms	`span`	#PCDATA
Blocks	`p`, `pre`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`	(#PCDATA | %Inlines;)*
Inlines	`a`, `span`, `sub`, `sup`, `tt`, `i`, `b`	(#PCDATA | %Inlines;)*
*Records**	`table`	(`tr`)*
Containers	`body`	(%Tables; | %Blocks;)*
	`li`	(%Tables; | %Blocks;)*
	`td`	(%Tables; | %Blocks;)*
	`th`	(%Tables; | %Blocks;)*
	`tr`	(`th` | `td`)*
Tables	`ul`	(`li`)*
	`ol`	(`li`)*
	`table`	(`tr`)*

Table 1. WIF Core

Understanding the role of each tag is immediate, as well as foreseeing a possible fragment of WIF. Actually there are some more specific features, like the internal structure of the `head` and `meta` elements for metadata, or `script` for behavior, and `style` for presentation. I have not mentioned them since they are less important for my purposes (as I said, I primarily focus on two out of five dimensions of the Pentaformat model).

However it is worth discussing briefly some elements and attributes dealing with semantics. WIF, in fact, is meant to be a standard language for semantic wikis as well, and needs constructs to express relations among elements, classes, instances, and so on. Then, in WIF 1.1 we proposed an extension for properties and relations, that exploits RDFa[AB06]. Consider, for instance, an in-line fragment indicating an e-mail. Semantic wikis mark that fragment, to search and re-use that semantic information. The example shows that fragment encoded in Semantic MediaWiki[VKV+06] and in WIF. The `meta` declaration is used to express the same information in WIF, hidden in the final page.

```
...contact me at [[email: diiorio@cs.unibo.it]]...

...contact me at <span class='email'>diiorio@cs.unibo.it</span>...

<meta property='email' content='diiorio@cs.unibo.it'/>
```

Semantic relations embedded in the document content can be expressed as well. The following examples show in-line fragments to indicate which is the license associated to a given document, according to the Semantic MediaWiki syntax and the WIF syntax:

```
this document is licensed under a
[[has\_license::http://creativecommons.org/licenses/by-nc/2.5/]
[Creative Commons Non-Commercial License]]

... this document is licensed under a
<a type='relation' rel='has\_license'
href='http://creativecommons.org/licenses/by-nc/2.5/'>
Creative Commons Non-Commercial License </a> ...
```

More than the actual details of WIF elements and attributes, it is important to remark their semantic expressivity. WIF is then a two-sided language: on the one hand, it is completely consistent with patterns (and provide users all advantages discussed about patterns and IML); on the other hand, it captures semantic information written by users as in-line fragments. Note that extensibility is one of the main principles behind the language: the set of tags proposed so far is meant to be extended, in order to meet specific requirements of specialized wikis.

Thus, contact points with IML are evident: IML relies on in-lined information and on a methodical use of the `@class` attributes, IML is modeled on few patterns and objects, IML is a core language extensible for specific needs and domains. WIF can be seen as a different instantiation of IML, characterized by some extra attributed and tags to express semantics. What is important, once again, is the overall pattern-based structure of documents, as well as the idea of embedding semantics within the text.

Chapter 9

Conclusions

The keyword to synthesize this thesis is 'minimality'. 'Minimality' because a small set of patterns is claimed to be enough to describe the actual content of any digital document. 'Minimality' because pattern-based objects have specific and constrained content models, and cannot be composed arbitrarily. 'Minimality' because few generic names and attributes were used in designing the core language of our applications, IML. The same acronym IML, which initially stood for 'IsaWiki Markup Language' because of the first system we worked on, and then became 'Intermediate Markup Language' to emphasize its role of superior standard format for automatic conversions, could be expanded as 'Intermediate Minimal Language'.

More than that minimality, the point is the opposition between the little of IML and the much of the applications based on that language. The principle behind the whole work can be reformulated in 'doing more with less': we actually used a limited set of constructs to build advanced and cross-domain applications. This 'tension' is also evident in the structure of the thesis, which is clearly divided in two parts. In the first five chapters I have described some design principles for descriptive documents, moving off the most discussed issues among markup and document-engineering communities. These principles resulted in a segmentation model based called 'Pentaformat', and some patterns able to express the actual structured content of any digital document. IML instantiates those abstract patterns into an actual markup language, readable and processable either by humans or machines. In the last three chapters I have presented some applications based on IML, which proved to be advantageous in heterogeneous scenarios: IsaWiki is a distributed editing environment where users can edit and customize documents regardless of data-format and access permissions, IsaPress is a tool for professional publishing which generates high-quality PDFs from raw sources, IsaLearning is a chain of tools to create high-quality learning objects from content written with ordinary word-processors, and WikiFactory is a stand-alone application which transforms an ontological description of a domain into a set of wiki pages for that domain, loadable on multiple wiki clones.

Besides internal and technical aspects, all these applications share motivations and goals: empowering authoring processes in order to let users achieve results which otherwise would be difficult or impossible. Reducing the burden of manual pagination for typesetters, skipping the manual intervention of editorial staff for e-leaning, automating the delivery of wiki content, and letting authors to re-use and publish their legacy material with little effort are all facets of an enhanced, in a sense *democratic*, vision of digital publishing.

In the last years, we have been witnessing at an increasingly presence of users in the World Wide Web, and in general in digital publishing. An increase and sophistication in the technologies and tools helped average users bring their own ideas, comments and opinions to the public and all other users. Weblogs, wikis, WYSIWYG editors, community spaces made possible informal, unconstrained and open contribution of texts and content by the inexpert users as well. This thesis followed the same path, and presented a pattern-based approach to implement applications that make users write and publish high-quality content without particular skills and tools.

I see a return to the roots of WWW and early publishing systems, a sort of 'ideal bridge' between what pioneers like Bush, Engelbart and Nelson foresaw many years ago, and what is common and quite natural today. The next step will be the Semantic Web, a powerful evolution aiming at making the WWW readable and interpretable by machines. Yet, that evolution is fascinating and interesting but there is still room to work on a publishing platform primarily addressed to humans. On the other hand, a Semantic Web still requires human interventions to insert data: those data will be later processed by machines, but at some point they have to be added by humans. It is worth spending time and resources in improving authoring processes and environments, in order to make possible create high-quality documents from any user's input. I mentioned the World Wide Web, since it is undoubtedly the most used platforms for digital publishing, but similar considerations can be extended to any other publishing scenario.

The key point is a real integration among different data formats and, in particular, the display of existing material in an integrated way with on-purpose content. This means allowing users to upload any resource as it is (produced by using well-known productivity tools), and then converting it into an internal format, so that it can be further converted into an high-quality output. That automatic conversion requires analysis of input files, and tools to extract and separate content and presentation. What my thesis did was *extending in a radical way* the principle of content/format separation, in order to propose a flexible and powerful *pattern-based segmentation model*, on which advanced applications of digital publishing can be designed and implemented.

References

[AB06] B. Adida and M. Birbeck. RDFa Primer 1.0 Embedding RDF in XHTML - W3C Wrding Draft. http://www.w3.org/TR/xhtml-rdfa-primer/, 2006.

[ACM02] S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. *Theor. Comput. Sci.*, 275(1-2):179–213, 2002.

[Ado99] Adobe. Adobe indesign. http://www.adobe.com/products/indesign/, 1999.

[AGV03] N. Amorosi, N. Gessa, and F. Vitali. Datatype- and namespace-aware DTDs. A minimal extension. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2003.

[AKM95] K. Andrews, F. Kappe, and H. Maurer. Serving information to the web with hyper-g. In *Proceedings of the Third International World-Wide Web conference on Technology, tools and applications*, pages 919–926, New York, NY, USA, 1995. Elsevier North-Holland, Inc.

[Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[All04] D. Allen. Word HTML Cleaner. http://www.textism.com/wordcleaner/, 2004.

[AMTW02] M. Aiello, C. Monz, L. Todoran, and M. Worring. Document understanding for a broad class of documents. *International Journal on Document Analysis and Recognition.*, 5:1–16, 2002.

[Apa01] Software Foundation Apache. Apache FOP. http://xmlgraphics.apache.org/fop/, 2001.

[Arc00] A. F. Arciniegas. Design patterns in xml applications: Part ii. http://www.xml.com/pub/a/2000/02/16/feature/index.html, 2000.

[Bau05] S. Bauman. TEI HORSEing Around. In *Proceedings of Extreme Markup Conference*, Montreal, Canada, 2005.

[BC87] K. Beck and W. Cunningham. Using Pattern Languages for Object-Oriented Programs. In *Proceedings of OOPSLA-87 - Workshop on the Specification and Design for Object-Oriented Programming.*, 1987.

[BCV99] L. Bompani, P. Ciancarini, and F. Vitali. Active documents in xml. *SIGWEB Newsl.*, 8(1):27–31, 1999.

[BG06] M. Buffa and F. Gandon. Sweetwiki: semantic web enabled technologies in wiki. In *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, pages 69–78, New York, NY, USA, 2006. ACM Press.

[BKGM+02] O. Buyukkokten, O. Kaljuvee, H. Garcia-Molina, A. Paepcke, and T. Winograd. Efficient web browsing on handheld devices using page and form summarization. *ACM Trans. Inf. Syst.*, 20(1):82–115, 2002.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. *The Semantic Web*. Scientific American, 2001.

[BM99] D. J. Birnbaum and D. A. Mundie. The problem of anomalous data: a transformational approach. *Markup Lang.*, 1(4):1–19, 1999.

[Boi01] B. Boiko. *Content management Bible*. John Wiley & Sons Inc., 2001.

[Bou99] N. O. Bouvin. Unifying strategies for web augmentation. In *HYPERTEXT '99: Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots*, pages 91–100, New York, NY, USA, 1999. ACM Press.

[Bri96] S. Bringsjord. Text is jottings plus procedure (jopp). `http://www.rpi.edu/~brings/SELPAP/monistel1/monistel1.html`, 1996.

[Bur05] R. Burget. Visual html document modeling for information extraction. In *First International Workshop on Representation and Analysis of Web Space*, pages 17–24. Faculty of Electrical Engineering and Computer Science, VSB-TU Ostrava, 2005.

[Bus45] V. Bush. As we may think. *Atlantic Monthly*, 1945.

[BVA+97] M. Bieber, F. Vitali, H. Ashman, V. Balasubramanian, and H. Oinas-Kukkonen. Fourth Generation Hypertext: Some Missing Links for the World Wide Web. *International Journal of Human-Computer Studies*, 47:31–265, 1997.

[BZI97] R. Brugger, A. Zramdini, and R. Ingold. Modeling documents for structure recognition using generalized n-grams. *icdar*, 00:56, 1997.

[CDRHH95] L. A. Carr, D. C. De Roure, W. Hall, and G. J. Hill. The Distributed Link Service: A Tool for Publishers, Authors and Readers. *The Web Journal*, 1(1):647–656, 1995.

[CFRV02] P. Ciancarini, F. Folli, D. Rossi, and F. Vitali. XLinkProxy: External Linkbases with XLink. In *Proceedings of the ACM symposium on Document Engineering*, pages 57–65, 2002.

[CGS02] C. Y. Chung, M. Gertz, and N. Sundaresan. Reverse engineering for web data: From visual to semantic structures. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 53, Washington, DC, USA, 2002. IEEE Computer Society.

[CL01] W. Cunningham and B. Leuf. *The Wiki way*. Addison-Wesley, New York, 2001.

[CM04] V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *Journal of ACM*, 51(5):731–779, 2004.

[CMZ03] Y. Chen, W. Ma, and H. Zhang. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 225–233, New York, NY, USA, 2003. ACM Press.

[Con87] TEI Consortium. The Text Encoding Initiative. `http://http://www.tei-c.org/`, 1987.

[CRD87] J. H. Coombs, A. H. Renear, and S. J. DeRose. Markup systems and the future of scholarly text processing. *Commun. ACM*, 30(11):933–947, 1987.

[CTK+05] D. Cederholm, . Tantek, R. Khare, R. King, and K. Marks. Microformats. `http://microformats.org/`, 2005.

[CVJ99] W. Chisholm, G. Vanderheiden, and I. Jacobs. Web Content Accessibility Guidelines - W3C Recommendation. http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/, 1999.

[CZS+01] J. Chen, B. Zhou, J. Shi, H. Zhang, and Q. Fengwu. Function-based object model towards website adaptation. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 587–596, New York, NY, USA, 2001. ACM Press.

[Dat81] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, New York, 1981.

[DD94] S. J. DeRose and D. Durand. *Making Hypermedia Work: A User's Guide to HyTime*. Kluwer Academic Publishers, Norwell, MA, USA, 1994.

[DDMR87] S. J. DeRose, D. Durand, E. Mylonas, and A. H. Renear. What is Text, Really? *Journal of Computing in Higher Education*, 1:3–26, February 1987.

[DeR97] S. J. DeRose. Further context for what is text, really?. *SIGDOC Asterisk J. Comput. Doc.*, 21(3):40–44, 1997.

[DeR04] S. J. DeRose. Markup Overlap: A Review and a Horse. In *Proceedings of Extreme Markup Conference*, Montreal, Canada, 2004.

[DHH+92] H. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins. Towards an integrated information environment with open hypermedia systems. In *ECHT '92: Proceedings of the ACM conference on Hypertext*, pages 181–190, New York, NY, USA, 1992. ACM Press.

[DHN06] J. Deriviere, T. Hamon, and A. Nazarenko. A scalable and distributed nlp architecture for web document annotation. In *Advances in Natural Language Processing*, volume 4139 of *Lecture Notes in Computer Science*, pages 56–67. Springer Verlag, 2006.

[Dic97] R. S. Dicks. Third commentary on "what is text really?". *SIGDOC Asterisk J. Comput. Doc.*, 21(3):36–39, 1997.

[DIFM+05] A. Di Iorio, A. A. Feliziani, S. Mirri, P. Salomoni, and F. Vitali. Simply Creating Accessible Learning Object. In *Proceedings of the Workshop on eLearning and Human-Computer Interaction in the 10th IFIP International Conference on HCI*, Roma, Italy, 2005.

[DIFM+06] A. Di Iorio, A. A. Feliziani, S. Mirri, P. Salomoni, and F. Vitali. Automatically Generating Accessible Learning Objects. *Journal on Educational and Technology*, 2006.

[DIFV06] A. Di Iorio, L. Furini, and F. Vitali. A Total-Fit Page-Breaking Algorithm with User-Defined Adjustment Strategies. In *Proceedings of the IST/SPIE Annual Symposyum on Electronic Imaging*, 2006.

[DIGV05] A. Di Iorio, D. Gubellini, and F. Vitali. Design Patterns for Descriptive Document Substructures. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2005.

[DIMV05] A. Di Iorio, G. Montemari, and F. Vitali. Beyond Proxies: Xlink Support in the Browser. In *Proceedings of the First International Conference on Internet Technologies and Applications*, Wrexham, Wales, 2005.

[DIPV06a] A. Di Iorio, V. Presutti, and F. Vitali. Automatic Deployment of Semantic Wikis: a Prototype. In *Proceedings of the First Workshop on Semantic Wikis*, 2006.

[DIPV06b] A. Di Iorio, V. Presutti, and F. Vitali. WikiFactory: an ontology-based application to deploy domain-oriented wikis. In *Proceedings of the European Semantic Web Conference*, 2006.

[DIV03] A. Di Iorio and F. Vitali. A Xanalogical Collaborative Editing Environment. In *Proceedings of the Second International Workshop on Web Document Analysis*, 2003.

[DIV04] A. Di Iorio and F. Vitali. Writing the Web. *Journal of Digital Information*, 5, may 2004.

[DIV05a] A. Di Iorio and F. Vitali. From the writable web to global editability. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 35–45, New York, NY, USA, 2005. ACM Press.

[DIV05b] A. Di Iorio and F. Vitali. Web Authoring: a Closed Case. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005.

[DIV06] A. Di Iorio and M. Voekel. Wif: Wiki interchage format. http://www.wikisym.org/wiki/index.php/WSR_3, 2006.

[DIVCV04] A. Di Iorio, E. Ventura Campori, and F. Vitali. Rule-based Structural Analysis of Web Pages. In Simone Marinai and Andreas Dengel Eds., editors, *Document Analysis VI*, volume 3163 of *Lecture Notes in Computer Science*, pages 425–437. Springer Verlag, 2004.

[DIZ06] A. Di Iorio and S. Zacchiroli. Constrained Wiki: an Oxymoron? In *ACM Symposyum on Wikis*, 2006.

[Dji80] T. A. V. Djik. *Macrostructures: An Interdisciplinary Study of Global Structures in Discourse, Interaction, and Cognition*. L. Erlbaum Associates, Hillsdale, NJ, 1980.

[DMD01] S. J. DeRose, E. Maler, and Orchard D. Xml linking language (xlink) version 1.0. `http://www.w3.org/TR/2001/REC-xlink-20010627`, 2001.

[DMHR03] D. Dubin, Sperberg-McQueen C. M., C. Huitfeldt, and A. H. Renear. A logic programming environment for document semantics and inference. *Literary and Linguistic Computing*, 2(18):215–234, 2003.

[DO02] P. Durusau and M. B. O'Donnell. Coming down from the trees: Next step in the evolution of markup? In *Late Breaking at Extreme Markup Conference*, Montreal, Canada, 2002.

[Dow03] K. Downey. Architectural design patterns for xml documents. `http://www.xml.com/pub/a/2003/03/26/patterns.html`, 2003.

[DWB02] L. M. Diaz, E. Wistner, and P. Buxmann. Inter-organizational document exchange: facing the conversion problem with xml. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1043–1047, New York, NY, USA, 2002. ACM Press.

[Fis91] J. L. Fisher. Logical structure descriptions of segmented document images. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 302–310, Saint-Malo, France, 1991.

[Fre98] D. Freitag. Toward general-purpose learning for information extraction. In *Proceedings of the 17th international conference on Computational linguistics*, pages 404–408, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[Gay01] G. Gay. ATutor - Learning Content Management System. http://www.atutor.ca/, 2001.

[GBS97] K. Gronbek, N. O. Bouvin, and L. Sloth. Designing dexter-based hypermedia services for the world wide web. In *HYPERTEXT '97: Proceedings of the eighth ACM conference on Hypertext*, pages 146–156, New York, NY, USA, 1997. ACM Press.

[GHHW01] B. Goodger, I. Hickson, D. Hyatt, and C. Waterson. Xml user interface language (xul). http://www.mozilla.org/projects/xul/, 2001.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, New York, 1994.

[GKDG03] S. Gupta, G. Kaiser, Neistadt D., and P. Grimm. Dom-based content extraction of html documents. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 207–214, New York, NY, USA, 2003. ACM Press.

[GM02] R. J. Glushko and T. McGrath. Document Engineering for e-Business. In *Proceedings of the ACM symposium on Document Engineering*, pages 42–48, McLean, Virginia, USA, 2002.

[GMMW03] P. Grosso, E. Maler, J. Marsh, and N. Walsh. XPointer element() Scheme. `http://www.w3.org/TR/xptr-element/`, 2003.

[Gol81] C. F. Goldfarb. A generalized approach to document markup. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pages 68–73, New York, NY, USA, 1981. ACM Press.

[Gol90] C. F. Goldfarb. *The SGML Handbook*. Clarendon Press, Oxford, 1990.

[GQ98] I. Graham and L. Quin. Introduction to XML Design Patterns. `http://www.utoronto.ca/ian/books/xmlbook/patterns.html`, 1998.

[GWF+99] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. Http extensions for distributed authoring - webdav. `http://www.ietf.org/rfc/rfc2518.txt`, 1999.

[Har99] E. Harold. *XML Bible*. IDG Books Worldwide, Foster City, CA, 1999.

[HB05] S. L. Henry and J. Brewe. Web Accessibility Initiative. http://www.w3.org/WAI/about.html, 2005.

[HC01] T. W. Hong and K. L. Clark. Using grammatical inference to automate information extraction from the web. In *PKDD '01: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 216–227, London, UK, 2001. Springer-Verlag.

[HH89] M. A. K. Halliday and R. Hasan. *Language, Context, and Text: Aspects of Language in a Social-semiotic Perspective*. Oxford University Press, Oxford, 1989.

[Hil02] T. Hillesund. Many Outputs Many Inputs: XML for Publishers and E-book Designers. *Journal of Digital Information*, 3, January 2002.

[IFL97] Study Group IFLA. Functional Requirements for Bibliographic Records. `http://www.ifla.org/VII/s13/frbr/frbr.pdf`, 1997.

[Jel05] R. Jelliffe. Schematron 1.5. `http://xml.ascc.net/schematron/`, 2005.

[JY98] A. K. Jain and B. Yu. Document representation and its application to page decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(3):294–308, 1998.

[KAK+00] E. Kaasinen, M. Aaltonen, J. Kolari, S. Melakoski, and T. Laakko. Two approaches to bringing internet services to wap devices. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 231–246, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[Kas98] B. Kasdorf. SGML and PDF – Why We Need Both. *The Journal of Electronic Publishing*, 3, June 1998.

[Kaw01] K. Kawaguchi. W3c xml schema made simple. `http://www.xml.com/pub/a/2001/06/06/schemasimple.html`, 2001.

[KDK00] S. Klink, A. Dengel, and T. Kieninger. Document structure analysis based on layout and textual features. In *Proceedings of the 4th IAPR International Workshop on Document Analysis Systems*, Brazil, 2000.

[Kha06] R. Khare. Microformats: The Next (Small) Thing on the Semantic Web? *Internet Computing, IEEE*, 10(1):68–75, 2006.

[KKPS01] J. Kahan, M. Koivunen, E. Prud'Hommeaux, and R. Swick. Annotea: An Open RDF Infrastructure for Shared Web Annotations. In *Proceedings of the Internation World Wide Web Conference*, Hong Kong, 2001.

[KLMM03] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: a machine learning approach. *Intelligent exploration of the web*, pages 275–287, 2003.

[KLT03] J. Kim, D. Le, and G. Thoma. Automated labeling algorithms for biomedical document images. In *Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 352–357, Orlando, FL, USA, 2003.

[KMMV02] M. Kovacevi, Diligenti M., Gori M., and Milutinovic V. Recognition of common areas in a web page using visual information: a possible application in a page classification. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 250, Washington, DC, USA, 2002. IEEE Computer Society.

[Kna03] F. C. Knabben. TCKEditor: the editor for Internet. `http://www.fckeditor.net/`, 2003.

[KNSV93] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan. Syntactic segmentation and labeling of digitized pages from technical journals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(7):737–747, 1993.

[KS04] J. Kraus and G. Spencer. Jotspot: the application wiki. http://www.jotspot.com/, 2004.

[Lai00] T. Lainevool. XML Design Patterns. `http://www.xmlpatterns.com/`, 2000.

[Lau98] S. S. Laurent. *XML: A Primer*. IDG Books Worldwide / MIS Press, Foster City, CA, 1998.

[LCC03] K. Lee, Y. Choy, and S. Cho. Logical structure analysis and generation for structured documents: A syntactic approach. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1277–1294, 2003.

[Lea04] Advanced Distributed Learning. SCORM: Sharable Content Object Reference Model. http://www.adlnet.gov/scorm/index.cfm, 2004.

[LGR05] J. Lumley, R. Gimson, and O. Rees. A framework for structure, layout & function in documents. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 32–41, New York, NY, USA, 2005. ACM Press.

[Liu04] A. Liu. Transcendental Data: Toward a Cultural History and Aesthetics of the New Encoded Discourse. *Critical Inquiry*, 1(31):49–84, 2004.

[MA02] D. Martin and H. Ashman. Goate: Xlink and beyond. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, pages 142–143, New York, NY, USA, 2002. ACM Press.

[Man02] M. Manske. Mediawiki. http://wikipedia.sourceforge.net/, 2002.

[MAR99] MARC. Marc Standards. `http://www.loc.gov/marc/`, 1999.

[MBCKH03] T. Miles-Board, L. Carr, S. Kampa, and W. Hall. Supporting management reporting: a writable web case study. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 234–243, New York, NY, USA, 2003. ACM Press.

[MLMK05] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. *ACM Trans. Inter. Tech.*, 5(4):660–704, 2005.

[MO04] J. Marsh and D. Orchard. Xml inclusions (xinclude) version 1.0. `http://www.w3.org/TR/xinclude/`, 2004.

[Mur00] M. Murata. Relax (REgular LAnguage description for Xml). http://www.xml.gr.jp/relax/, 2000.

[MYTR03] S. Mukherjee, G. Yang, W. Tan, and I. V. Ramakrishnan. Automatic discovery of semantic structures in html documents. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 245, Washington, DC, USA, 2003. IEEE Computer Society.

[MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 122–133, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[Nel87] T. H. Nelson. *Literary Machines*. Mindful Press, Sausalito, 1987.

[Nil98] M. Nilsson. ID3: The audience is informed. `http://www.id3.org/`, 1998.

[NIS01] NISO. The Dublin Core Metadata Initiative. `http://www.niso.org/standards/resources/Z39-85.pdf`, 2001.

[NS95] D. Niyogi and S. N. Srihari. Knowledge-based derivation of document logical structure. *icdar*, 01:472, 1995.

[NSO03] T. Nanno, S. Saito, and M. Okumura. Structuring Web pages based on Repetition of Element. In *Proceedings of the Second International Workshop on Web Document Analysis*, 2003.

[NSV92] G. Nagy, S. Seth, and M. Viswanathan. A prototype document image analysis system for technical journals. *Computer*, 25(7):10–22, 1992.

[Oba02] D. Obasanjo. W3c xml schema design patterns: Avoiding complexity. `http://www.xml.com/pub/a/2002/11/20/schemas.html`, 2002.

[O'G93] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1162–1173, 1993.

[Ore05] E. Oren. Semperwiki: a semantic personal wiki. In *Proc. of 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure*, Galway, Ireland, 2005.

[OV06] E. Oren and M. Voekel. Towards a wiki interchange format (wif). In *Proceedings of the First Workshop on Semantic Wikis*, 2006.

[Par99] W. J. Pardi. *XML in Action*. Microsoft Press, Redmond, WA, 1999.

[PHLM01] G. Penn, J. Hu, H. Luo, and R. McDonald. Flexible web document analysis for delivery to narrow-bandwidth devices. In *ICDAR '01: Proceedings of the Sixth International Conference on Document Analysis and Recognition*, page 1074, Washington, DC, USA, 2001. IEEE Computer Society.

[Pie01] W. Piez. Beyond the 'descriptive vs. procedural' distinction. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2001.

[Pie05] W. Piez. Format and Content: Should they be separated? Can they be?: With a counter-example.. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2005.

[pre04] PREMIS: PREservation Metatada Implementation Strategies. `http://www.oclc.org/research/projects/pmwg/`, 2004.

[Pro02] W. Provost. Structural patterns in xml. `http://www.xml.com/pub/a/2002/09/04/strucpatterns.html`, 2002.

[Qui96] L. Quin. Suggestive Markup: Explicit Relationships in Descriptive and Prescriptive DTDs. In *Proceedings of the SGML 96 Conference*, pages 119–126, Boston, MA, USA, 1996.

[Qui06] L. Quin. Microformats: Contaminants or Ingredients? Introducing MDL and Asking Questions. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2006.

[Ray99] E. S. Raymond. *The Cathedral & the Bazaar (Hardback) Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly Editions, 1999.

[RDM96] A. H. Renear, D. Durand, and E. Mylonas. Refining our Notion of What Text Really is: The Problem of Overlapping Hierarchies. *Research in Humanities Computing*, 1996.

[RDSM02] A. H. Renear, D. Dubin, and C. M. Sperberg-McQueen. Towards a Semantics for XML Markup. In *Proceedings of the ACM symposium on Document Engineering*, pages 119–126, 2002.

[Ren01] A. H. Renear. The Descriptive/Procedural Distinction is Flawed. *Markup Languages: Theory and Practice*, 4(2):411–420, 2001.

[RTW96] D. R. Raymond, F. W. Tompa, and D. Wood. From data representation to data model Meta-semantic issues in the evolution of SGML. *Computer Standards and Interfaces*, 1(18):2536, 1996.

[SCMV04] C. Sacerdoti Coen, P. Marinelli, and F. Vitali. Schemapath, a minimal extension to xml schema for conditional constraints. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 164–174, New York, NY, USA, 2004. ACM Press.

[SMB97] C. M. Sperberg-McQueen and L. Burnard. A Gentle Introduction to SGML. *In Guidelines for Electronic Text Encoding and Interchange*, pages 13–36, 1997.

[SMB00] C. M. Sperberg-McQueen and L. Burnard. A Gentle Introduction to XML. *In Guidelines for Electronic Text Encoding and Interchange*, 2000.

[SMB02] C. M. Sperberg-McQueen and L. Burnard. *TEI Guidelines for Electronic Text Encoding and Interchange*. University of Oxford Press, Oxford, MA, 2002.

[SMHR00] C. M. Sperberg-McQueen, C. Huitfeldt, and A. H. Renear. Meaning and interpretation of markup. *Markup Languages: Theory and Practice*, 3(2):215–234, 2000.

[SMRK03] A. Song Mao, A. Rosenfeld, and T. Kanungob. Document Structure Analysis Algorithms: A Literature Survey. In *Proceedings of the IST/SPIE Annual Symposyum on Electronic Imaging*, 2003.

[Sou05] A. Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, 20(5):87–91, 2005.

[SSJC06] C. Sauer, C. Smith, J. Jalkanen, and W. Cunningham. Wiki creole. http://www.wikicreole.org/, 2006.

[ST01] A. Salminin and F. Tompa. Requirements for XML document database systems. In *Proceedings of the ACM symposium on Document Engineering*, pages 85–94, 2001.

[SW01] L. Sanger and J. Wales. WikipediA, The Free Encyclopedia. `http://www.wikipedia.org/`, 2001.

[SYG99] S. N. Srihari, W. Yang, and V. Govindaraju. Information theoretic analysis of postal address fields for automatic address interpretation. In *ICDAR '99: Proceedings of the Fifth International Conference on Document Analysis and Recognition*, page 309, Washington, DC, USA, 1999. IEEE Computer Society.

[SZM+02] M. C. Schraefel, Y. Zhu, D. Modjeska, D. Wigdor, and S. Zhao. Hunter Gatherer: Interaction Support for the Creation and Management of Within-web-page Collections. In *Proceedings of the Internation World Wide Web Conference*, pages 172–181, Honolulu, Hawaii, USA, 2002.

[TA90] S. Tsujimoto and H. Asada. Understanding multi-articled documents. In *Proceedings of the 10 th International Conference on Pattern Recognition*, pages 16–21, Atlantic City, NJ, 1990. IEEE Computer Society.

[TBMM01] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. `http://www.w3.org/TR/xmlschema-1/`, 2001.

[TP02] J. Tennison and W. Piez. The Layered Markup and Annotation Language (LMNL). In *Late Breaking at Extreme Markup Conference*, Montreal, Canada, 2002.

[Usd02] B. T. Usdin. When 'it doesn't matter' Means 'it matters'. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2002.

[Vit03] F. Vitali. Creating Sophisticated Web Sites using Well-known Interfaces. In *Proceedings of the HCI International 2003 Conference*, Crete, Greece, 2003.

[VKV+06] M. Volkel, M. Krotzsch, D. Vrandecic, H. Haller, and R. Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*, MAY 2006.

[W3C01] W3C. Amaya. `http://www.w3.org/Amaya/`, 2001.

[Wal99] N. Walsh. DocBook: The Definitive Guide. `http://www.docbook.org/`, 1999.

[Wal02] N. Walsh. XML: One Input Many Outputs: a response to Hillesund. *Journal of Digital Information*, 3, January 2002.

[Wil02] S. Wilmott. The Dichotomy of Markup Languages. In *Proceedings of the Extreme Markup Conference*, Montreal, Canada, 2002.

[Yat06] M. Yatsu. LesserWiki: an ajax notebook. `http://lesserwiki.org/`, 2006.

[YZ01] Y. Yang and H. Zhang. Html page analysis based on visual clues. In *ICDAR '01: Proceedings of the Sixth International Conference on Document Analysis and Recognition*, page 859, Washington, DC, USA, 2001. IEEE Computer Society.