Dealing with structural patterns of XML documents

Angelo Di Iorio

Silvio Peroni

Francesco Poggi

Fabio Vitali

Keywords: XML, descriptive markup, document visualisation, ontology, pattern recognition, structural patterns

Author note

Angelo Di Iorio, Department of Computer Science and Engineering, University of Bologna, Bologna, Italy,
email: diiorio@cs.unibo.it

Silvio Peroni, Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
email: essepuntato@cs.unibo.it [Corresponding Author]

Francesco Poggi, Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
email: fpoggi@cs.unibo.it

Fabio Vitali, Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
email: fabio@cs.unibo.it

ABSTRACT

Evaluating collections of XML documents without paying attention to the schema they were written in may give interesting insights about the expected characteristics of a markup language, as well as bout any regularity that may span across vocabularies and languages, and that are more fundamental and frequent than plain content models. In this paper we explore the idea of structural patterns in XML vocabularies, by examining the characteristics of elements as they are used, rather than as they are defined.

We introduce from the ground up a formal theory of eight plus three structural patterns for XML elements, and verify their identifiability in a number of different XML vocabularies. The satisfying results have allowed the creation of visualization and content extraction tools that are completely independent of the schema and without any previous knowledge of the semantics and organization of the XML vocabulary of the documents.

INTRODUCTION

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

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

The focus of this work is on the analysis and design of XML vocabularies regardless of their schema. This perspective differs from other works on XML validation since, instead of looking at the expressivity of validation languages in defining element labels and imposing constraints on their use and positions, this paper investigates document instances in order to derive classes of elements persisting across documents and distilling the conceptualization of the documents and their components. In fact, our focus is on *XML structural patterns*.

Patterns are useful in all stages of the documents' lifecycle: they can be *guidelines* for creating well-engineered documents and vocabularies, *rules* to extract structural components from legacy documents, *indicators* to study to what extent documents share design principles and community guidelines. The novel and challenging aspect of our approach is that we build our patterns from the ground up, defining a small set of eight fundamental patterns, plus three important subpatterns and some composition rules, that are sufficient to express what authors of documents and schemas most frequently need.

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

The identification of these sub-schemas is not a goal we have, since compliance to it is often spontaneous and often unrecognised by the very authors. Yet, *in practice* authors do tend to associate patterns to the elements of a vocabulary and use them accordingly, although with some exceptions we discuss in the paper. One straightforward application of the pattern theory is that it is possible to build useful applications (such as a visualizer and a index builder) automatically and without any previous knowledge about the vocabulary used, as we show with a little prototype at the end of the paper.

This paper extends several of our previous works ((Di Iorio, Peroni, Poggi & Vitali, 2012) (Di Iorio, Gubellini & Vitali, 2005) (Dattolo, Di Iorio, Duca, Feliziani & Vitali, 2007)), and gives a detailed and formal description of our model, completing the theory and reformulating the relations between patterns (and groups of patterns) in a more rigorous, reliable and manageable way. Furthermore it performs systematic tests of pattern compliance on eight very different vocabularies for a total of more than 1100 different documents.

The rest of the paper is then organised as follows. In Section "Structural patterns" we give an overview of structural patterns and existing literature on the topic; details of our theory and a formal description of it are provided in Section "A theory of patterns". In Section "Determining structural patterns in documents" we discuss our algorithm for the automatic recognition of structural patterns. In Section "Checking patterns on real-world documents" we present our studies on patterns' adoption in eight different XML vocabularies. In Section "Applying patterns to visualization and navigation: PViewer" we discuss a few sample tools that use the pattern theory to generate useful functionalities, such as visualization and indexing, before concluding in Section "Conclusions".

STRUCTURAL PATTERNS

The idea of using patterns to produce reusable and high-quality assets is not new in the literature. Software engineers (Gamma, Helm, Johnson & Vlissides, 1994), architects and designers often use – or indeed *reuse* – patterns to handle

problems that occur over and over again. Patterns have also been studied to modularize and customize web ontologies (Presutti & Gangemi, 2008) (Yeh, Wu & M. J., 2008). They guarantee the flexibility and maintainability of concepts and solutions in several heterogeneous scenarios.

We have been investigating patterns for XML documents (e.g. (Di Iorio et al., 2012) (Dattolo et al., 2007) (Di Iorio et al., 2005)) to understand how the structure of digital documents can be segmented into smaller components, which can be addressed independently and manipulated for different purposes. Instead of defining a large number of complex and diversified structures, we found a small number of *structural patterns* that are sufficient to express what most users need.

The two main characterizing aspects of such set of patterns are:

- *orthogonality* – each pattern has a specific goal and fits a specific context. The orthogonality between patterns makes it possible to associate a single pattern to each of the most common situations in document design. Conversely, for every situation a designer encounters in the creation of a new markup language, the corresponding pattern is immediately selectable and applicable;
- *assemblability* – each pattern can be used only in some contexts within other patterns. Far from being a limitation, this strictness provides expressiveness and non-ambiguity in the patterns. By limiting the possible choices, patterns prevent the creation of uncontrolled and misleading content structures.

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

The idea of defining document and element meta-structures is actually rooted in the early days of markup languages, such as through Architectural Forms as proposed for SGML and HyTime (e.g., see (DeRose & Durand, 1994)). Architectural Forms allow designers to express semantic information about specific instances of an element or about all elements of a given type. This is possible by extending the set of attributes and by setting some of their values appropriately. Such information does not impact the basic processing and integrity of the document but makes it possible to describe meta-structures and to define the semantic role of the elements. Constraints between meta-structures can also be described explicitly.

In particular, the automatic recognition of structural patterns can help in the visualization of XML documents. Given a set of schema-homogeneous XML documents, our idea is to identify markup elements and their related compliant structural patterns knowing neither the implicit semantics of markup elements nor any presentational stylesheet associate with the schema.

Some literature has recently come out about the characterization and identification of structural patterns of text documents. For instance, Tannier *et al*. (Tannier, Girardot & Mathieu, 2005), starting from previous works by Lini *et al*. (Lini, Lombardini, Paoli, Colazzo & Sartiani, 2001) and Colazzo *et al*. (Colazzo et al., 2002), describe an algorithm to assign each XML element in a document to one of three different categories: *hard tag*, *soft tag* and *jump tag*. *Hard tags* are elements that are commonly used to structure the document content in different blocks and usually "interrupt the linearity of a text": for instance, in the DocBook vocabulary (Walsh, 2010), they correspond to, among others, *para*, *section*, *table* etc. *Soft tags* are the elements that identify significant text fragments and are "transparent while reading the text"*:* they are mostly inline elements carrying presentation rules (e.g., in DocBook, *emphasis*, *link*, *xref*, etc.). Finally, *jump tags* are elements that are logically "detached from the surrounding text" and that give access to related information – e.g., in DocBook, *footnote*, *comment*, *tip*, etc. Tannier *et al*. also introduce algorithms to assign XML elements to these categories by means of NLP tools. This classification is rather interesting, in that it provides a justification for the identification of the classes, but it is a little coarse for our purposes, ignoring empty elements and failing to distinguish higher level and lower level hard tags (i.e., those containing other tags but not text from those that never contain text). In Section "A theory of patterns" we introduce a finer distinction, but it is interesting to note that the *soft tags* category is very close to our *Inline* pattern, the *jump tags* is similar to our *Popup* pattern, but their *hard tags* group comprises both our *Block* and *Container* pattern.

Zou *et al*. (Zou, Le & Thoma, 2007) categorise HTML elements as belonging to two classes only: *inline* and *line-break* tags. Inline elements all those that do not provide horizontal breaks in the visualisation of documents – e.g., *em*, *a*, *strong* and *q*, while line-break elements are those that do so – e.g., *p*, *div*, *ul*, *table* and *blockquote*. Based on this categorisation and a Hidden Markov Model the authors try to identify the structural role (e.g., title, author, affiliation, abstract, etc.) of textual fragments of medical journal articles expressed as HTML pages. Higher-level structural roles (e.g., *div* elements used as section separators) are not discussed nor identified; similarly, out-of-flow elements (corresponding to jump tags in (Tannier et al., 2005) and to the *Popup* pattern in our classification) do not really exist as such in HTML and therefore are clearly not identified.

Koh *et al*. (Koh, Caruso, Kerne & Gutierrez-Osuna, 2007) identify text fragments and images of documents that can act

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

Similarly, Vitali *et al*. implemented a rule-based system for the analysis of regularities and structures within web pages (Vitali, Di Iorio & Campori, 2004). The system seeks patterns in the HTML code of a page and labels the components of that page according to these patterns. One key aspect of the system is its extensibility. There is in fact a strong distinction between the rule engine and the actual patterns, which are declaratively expressed through XPath expressions in a custom XML vocabulary. Authors define an initial set of patterns to recognize, for instance: table cells, editable regions, navigational elements and annotated non-navigational text fragments.

Finally, Georg *et al*. (Georg & Jaulent, 2007) introduce an NLP approach to the automatic processing of medical texts such as clinical guidelines, in order to identify linguistic patterns that support the identification of the markup structure of documents. This approach justifies the development of a system for the automatic visualisation and presentation of unstructured documents. In a more recent paper (Georg, Hernault, Cavazza, Prendinger & Ishizuka, 2009) Georg *et al*. illustrate an extension of such a work in which they introduce an improved version of their approach.

A THEORY OF PATTERNS

In this section we introduce a novel method to address document patterns, which generates, in our view, a systematic collection of interesting patterns by specifying a few meta-structures and some precise rules to combine them. Patterns are organized around two orthogonal dimensions: their *content model* and their *context*. The *content model* indicates the structures or text nodes (possibly intermixed with each other) that an element can contain as well as their composition rules, while the *context* indicates the elements in which that element can appear. There is a strong relation between content models and contexts. If an element *A* can contain an element *B*, in fact, two relations hold: *B* belongs to the content model of *A*, and *A* belongs to the context of *B*. This constitutes the basis for our whole theory. Order relations of the elements of a content model should also be considered, and an example will be given in section "Specialisations of the Container pattern".

The basic layer, in fact, consists of precise definitions of some properties of markup elements and their content. The whole theory presented in the following sub-sections is formally defined through description logic formulas (Horrocks, Patel-Schneider, McGuinness & Welty, 2007) (Krotzsch, Simancik & Horrocks, 2011) and has been implemented as an OWL ontology (Motik, Patel-Schneider & Parsia, 2012) available at http://www.essepuntato.it/2008/12/pattern. The choice of description logic (DL) was mainly due to the application environment in which we further process such meta-structures. As we discuss later in this paper, in fact, we have developed an engine that recognizes these patterns by exploiting Semantic Web technologies and OWL-DL reasoning capabilities, which work on axioms of description logic. The transparent integration with Semantic Web data was another key factor for using OWL-DL, which allows the combination of the identification of meta-structures, as performed through our ontology, with other sources of information so as to validate content at different levels of abstraction and to perform sophisticated queries and analyses, such as studying peculiarities of the documents and their editing processes.

Basic properties of content models and contexts

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

We define *Textual* the class of markup elements that *can have textual content* in their content models and *NonTextual* (clearly disjoint with Textual) the class of elements that cannot. These two classes are disjointed. We also define *Structured* the class of markup elements that can *contain other markup elements*, and *NonStructured* as the class of elements that cannot. These two classes are disjoint[1].

---

[1] The pattern theory is introduced by means of description logic (DL) formulas. We briefly introduce the DL notation in order to help readers in reading the formalities of our theory: "⊤" and "⊥" refer to the *top* and *bottom concepts*, respectively; "⊑" expresses *concept inclusion*; "≡" expresses *concept equivalence*; "¬", "⊔" and "⊓" express *negation*, *disjunction* (i.e. union) and *complement* respectively; "-" expresses the *inverse role*, while "∃" and "∀" express *existential* and *universal restrictions*; "≤" express the *at-most restriction* ("≤nR" refers to the set of individuals that are related, through a relation *R*, to at most *n* of other individuals); ":" expresses a *value restriction*, ("R:v" is the set of individuals that are related, through a particular relation *R*, to a specific value*)*; for more details, see (Horrocks, Patel-Schneider, McGuinness & Welty, 2007) and (Krotzsch, Simancik & Horrocks, 2011).

$\mathtt{Textual} \sqsubseteq \top$

$\mathtt{NonTextual} \sqsubseteq \top$

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

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

$\mathtt{Structured} \sqsubseteq \top$

$\mathtt{NonStructured} \sqsubseteq \top$

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

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

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

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

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

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

- class *Marker*. Individual of this class can contain neither text nodes nor elements.
- class *Flat*. Individual of this class can contain text nodes but no elements;
- class *Bucket*. Individual of this class can contain other elements but no text nodes;
- class *Mixed*. Individuals of this class can contain other elements as well as text nodes;

These classes are defined as follows and shown together with their superclasses in Figure 1.

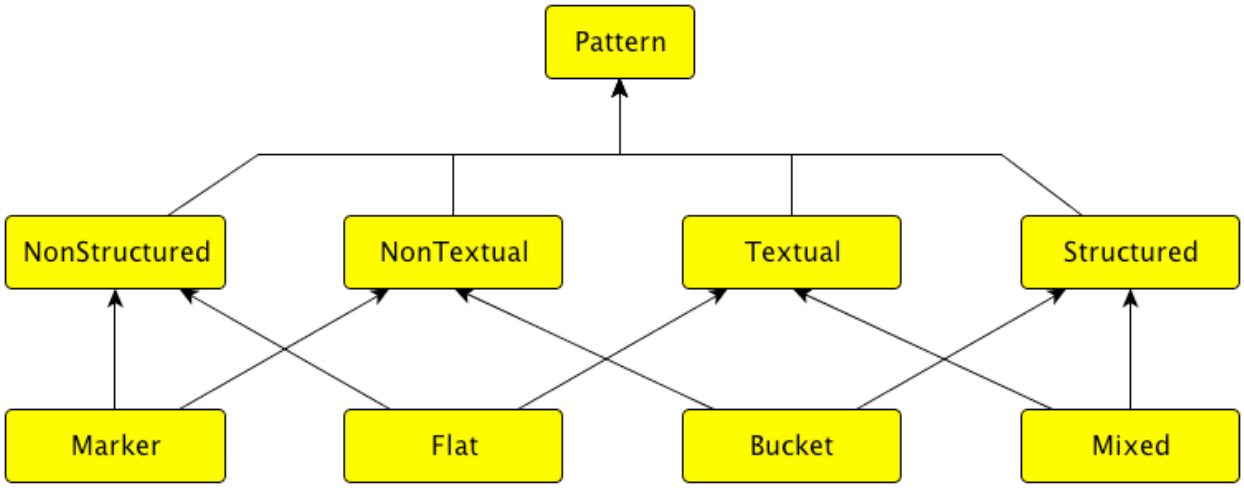

Figure 1. The abstract classes defining the hierarchical structure structural patterns are derived from. The arrows indicate sub-class relationships between patterns (e.g. Mixed is sub-class of Structured).

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

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

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

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

The behaviour of the content models can be fully described by these classes. Contexts can be characterized in a similar way, with the only important difference that, since each element clearly appears only in a content model that accepts elements, i.e., in a structured content model, the context of an element can only be either *Mixed* or *Bucket,* depending on whether it contains text or not.

Structural patterns

The combination of the possible content models (i.e. *Marker*, *Flat*, *Bucket*, *Mixed*) and contexts (i.e. *Mixed* and *Bucket*) bring to the identification of exactly eight different patterns. The abstract classes of our ontology, in fact, can be specialized into eight concrete patterns as shown in Figure 2. Notice that, for each pair of patterns derived from the same abstract class, the left one has a *Mixed* context and the right one a *Bucket*.

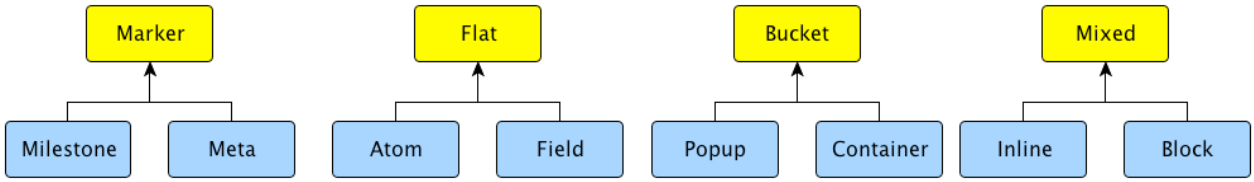

Figure 2. The eight concrete patterns derived from the abstract classes of our ontology. The arrows indicate sub-class relationships between patterns.

The relations between patterns, their content models and their contexts are also highlighted in Figure 3.

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

Figure 3. The eight patterns classified according to the particular content model and context they have.

Table 1 instead summarizes all patterns giving a brief description of their goal and some examples from HTML and DocBook vocabulary (Walsh, 2010).

Table 1. The eight structural patterns for descriptive documents.

Pattern	**Description**	**HTML**	**DocBook**
Milestone	Any content-less structure (but data could be specified in attributes) that is allowed in a mixed content structure but not in a container. The pattern is meant to represent locations within the text content that are relevant for any reason.	br	xref, co
Meta	Any content-less structure (but data could be specified in attributes) that is allowed in a container but not in a mixed content structure. The pattern is meant to represent metadata elements that assert things about the document, but are disconnected to its actual text content.	meta, cols, colspan, area	imagedata, colspec
Atom	Any simple box of text, without internal substructures (simple content) that is allowed in a mixed content structure but not in a container.	-	email, code

Field	Any simple box of text, without internal substructures (simple content) that is allowed in a container but not in a mixed content structure.	title	pubdate, publishername
Popup	Any structure that, while still not allowing text content inside itself, is nonetheless found in a mixed content context. The pattern is meant to represent complex substructures that interrupt but do not break the main flow of the text, such as footnotes.	-	footnote, tip
Container	Any container of a sequence of other substructures and that does not directly contain text. The pattern is meant to represent higher document structures that give shape and organization to a text document, but do not directly include the content of the document.	html, body, table, map	bibliography, preface
Inline	Any container of text and other substructures, including (even recursively) other inline elements. The pattern is meant to represent inline-level styles such as bold, italic, etc.	b, i, a, span	emphasis
Block	Any container of text and other substructures except for (even recursively) other block elements. The pattern is meant to represent block-level elements such as paragraphs.	p, div, address	para, caption

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

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

The elements of the *Milestone* pattern are empty and can only be contained within mixed elements (and consequently they also cannot be used as root elements of documents). The formal definition of this class is as follows:

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

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

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

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

formalized as follows:

```
Meta ≡ Marker ⊓ ∀isContainedBy.Bucket
Meta ⊓ Milestone ⊑ ⊥
```

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

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

```
Atom ≡ Flat ⊓ ∀isContainedBy.Mixed
Atom ⊑ ∃isContainedBy.Mixed
```

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

```
Field ≡ Flat ⊓ ∀isContainedBy.Bucket
Field ⊓ Atom ⊑ ⊥
```

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

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

```
Popup ≡ Bucket ⊓ ∀isContainedBy.Mixed
Popup ⊑
  ∀contains.(Container ⊔ Field ⊔ Meta ⊔ Block) ⊓
  ∃isContainedBy.Mixed
```

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

The sibling pattern *Container* concerns the structural organization of a document. Elements following this pattern contain no textual content and contain only elements compliant with the patterns *Meta*, *Field*, *Block* and *Container*. *Container* shares the same content model of *Popup* but they may be contained only in bucket elements, which makes these classes disjoint. Its formalisation is as follows:

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

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

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

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

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

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

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

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

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

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

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

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

$$\mathtt{Inline} \sqsubseteq$$

$\forall$contains.(Inline $\sqcup$ Atom $\sqcup$ Milestone $\sqcup$ Popup) $\sqcap$

$\exists$isContainedBy.Mixed

Block $\sqcap$ Inline $\sqsubseteq$ $\bot$

Specialisations of the Container pattern

Container has a very general definition. It is thus useful to define subclasses that describe situations all ascribable to the Container pattern, but that have a typicality worth of their own pattern. The class *Container*, in fact, can be specialized into at least three sub-classes (*HeadedContainer*, *Table* and *Record*) as shown in Figure 4.

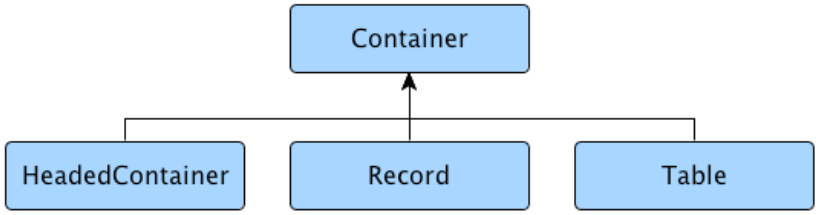

Figure 4. The three subclasses of the class Container.

Table 2 describes briefly these patterns and reports some examples in HTML and DocBook. Their formalisation follows in this section.

Table 2. The three sub-patterns of the Container pattern.

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

While the content model of structured elements (i.e. mixed and bucket elements) can contain any kind of optional and repeatable selection of elements, we need to be able to define some restrictions to their element repeatability so as to characterise the specialisation of the *Container* pattern. We thus define the boolean properties *canContainHomonymousElements,* true if the element can contain elements that share the same name[2], and *canContainHeteronymousElements*, true if an element can contain elements with different names. In addition, we define *containsAsHeader* as a sub-property of *contains* to specify when a structured-based element contains *header* elements.

```
∃canContainHomonymousElements.⊤ ⊑ Structured
⊤ ⊑ ≤1canContainHomonymousElements
∃canContainHeteronymousElements.⊤ ⊑ Structured
⊤ ⊑ ≤1canContainHeteronymousElements
containsAsHeader ⊑ contains
```

Through these new properties, we can define, among many, three subtypes of the *Container* pattern that we found particularly useful. For instance, the pattern *Record* captures the characteristics (typical of database records) of having many differently named elements with no repetitions. As such, its element can only contain heteronymous and non-repeatable elements, as in the following axioms:

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

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

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

```
Table ≡
  Container ⊓
```

2 By *name* we mean the pair *(namespace, general identifier)* of XML elements.

```
  canContainHomonymousElements:true ⊓
  canContainHeteronymousElements:false
```

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

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

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

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

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

```
Table ⊓ Record ⊑ ⊥
HeadedContainer ⊓ Record ⊑ ⊥
HeadedContainer ⊓ Table ⊑ ⊥
```

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

Assigning patterns to documents

The rules introduced in the previous sections allow us to assign one specific pattern to each element of a document, by analysing its local content model and context. Let us introduce an example to clarify this issue[3]:

```
<section>
  <title>Available physical types</title>
```

3 The document is taken from http://www.balisage.net/Proceedings/vol5/xml/Rennau01/BalisageVol5-Rennau01.xml. Some content has been removed for the sake of clarity.

```
  <para> As the results of a query execution […] </para>
  <para>  <emphasis  role="ital">Note:</emphasis>  The  preceding  subsection
introduced the notion of physical types […] </para>
  <table>
    <caption>
      <para>
        <emphasis  role="bold"><emphasis  role="ital">Summary  of  Java  types
delivered by XQJ.</emphasis></emphasis>
      </para>
    </caption>
    <col align="left" valign="top" span="1"/>
    <col align="left" valign="top" span="1"/>
    <thead>
      <tr valign="top">
       <th align="left" valign="top">category</th>
       <th align="left" valign="top">types</th>
      </tr>
    </thead>
    <tbody>
      <tr valign="top">
       <td><emphasis role="bold">atomic types</emphasis></td>
       <td>Boolean, BigDecimal, BigInteger, […] </td>
      </tr>
      <tr valign="top">
       <td><emphasis role="bold">node types</emphasis></td>
       <td>Document, Element, Attr, […] </td>
      </tr>
    </tbody>
  </table>
  <para> … </para>
  …
</section>
```

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

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

Coherency and pattern shifts

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

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

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

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

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

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

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

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

All the admissible shifts are indicated with arrows in Figure 5. They allow us to change the content model of an element without changing its context. Shifts that go in the opposite direction are not valid, as they would lose information. For

instance, they would not consider the presence of text when shifting back from Bucket to Marker.

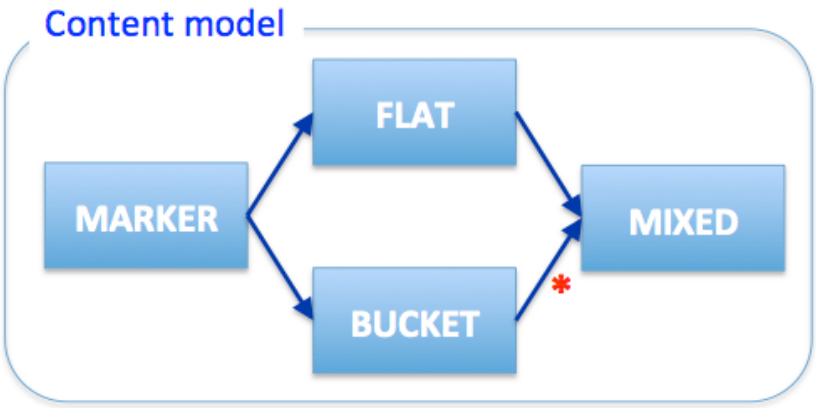

Figure 5. All the acceptable shifts. The asterisk as label of the arrow between Bucket and Mixed refers to a particular case of shifts, called *shifty-shifts*, which are still possible even if they change drastically the context (and, thus, the pattern) of all the elements contained by the shifted one.

We define the coherency obtained through shifts as follows.

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

There is a particular kind of shifts called *shifty-shifts*, labelled with an asterisk in Figure 5, which is particularly delicate to address. A shifty-shift from Bucket to Mixed applied on an element *E* actually changes the context of all the children of *E*, and may change radically the structure of a document. It is the case of the elements *td* in the previous example, which can be shifted to Blocks and, thus, modify the nature of all the elements *para* they contain from Block to Inline. Were this to happen to an element in the higher levels of the document hierarchy, even within a single document of a large set, it would completely disrupt the nature of all documents, whereby, for instance, the document element becomes the only Block and everything inside it becomes an indistinguishable Inline.

Pattern schemes and partitions

Once we have identified the patterns of the several elements of a set of XML documents, we can group all the assignments according to *pattern schemes*:

Definition 4: pattern scheme. Given a finite set of XML documents D, a *pattern scheme* S_D is the set of all the assignments *element-pattern* identified in D.

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

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

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

DETERMINING STRUCTURAL PATTERNS IN DOCUMENTS

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

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

Identification of potential content models and contexts. In this step we identify which of the four possible content models – empty (i.e. Marker), only text (i.e. Flat), only element (i.e. Bucket), both text and element (i.e. Mixed) – can be associated to each element, and thereby to identify the context for each of its children. This is the place where shifts

come into play: whenever different occurrences of the same element appear to have different content models it tries to generalize them in a single one.

Pattern assignment. Next, a pattern is assigned to each element instance, starting form the content model and context identified in the previous step. This is a direct application of the rules summarized in Figure 3.

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

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

The opposite operation, ***container generalization*****,** can be performed as a step towards global coherency: generalizing *Record*s, *Table*s, *HeadedContainer*s into simple *Container*s. Consider, for instance, the case of an element recognized as *record* in some documents and *table* in others. That might happen because the element is meant to collect heterogeneous information but, in some cases, it contains several different elements with no repetitions while in others it only contains only one element (and is therefore recognized as *Table*). It is therefore appropriate to generalize these patterns as *Containers*. Of course generalizing to containers is in a way to surrender the specialization of the containers and accept that some containers simply cannot be generalized. Fortunately, the recourse to this operation has been restricted to just a few well-justified situations.

Validation. The last step consists in verifying whether the associations between elements and patterns are valid. This is actually an optional step, just added to improve reliability and to double-check the final output. As described in (Di Iorio, Peroni & Vitali, 2011), this test can be performed easily using the technologies of the Semantic Web in three

steps:

- converting the XML document given in input in EARMARK (Di Iorio, Peroni & Vitali, 2011) (a version of the conversion tool is available online[4]);
- associating the previously calculated pattern to each element (through a *rdf:type* assertion);
- launching a reasoner to check if the Pattern Ontology[5] with these added assertions (the EARMARK document and the pattern associations) is consistent (all the pattern constraints hold) or not (there are some errors when assigning patterns to elements).

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

CHECKING PATTERNS ON REAL-WORLD DOCUMENTS

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

For instance, the development of vocabularies used by large communities such as TEI (Text Encoding Initiative Consortium, 2005) and DocBook (Walsh, 2010) has been (and still is) a long process that had to deal with complex constraints: schema designers are often required to capture all requirements of their prospective users, covering very heterogeneous situations and foreseeing any potential validation mismatch or misinterpretations. These difficulties conjure to produce overly rich and complex schemas, that require time and effort to be fully understood and applied, but that have an extremely simple core that had to be widened and made more complicated by special needs.

We rather propose to analyze the characteristics that emerge from real markup documents, not preventing any peculiar

4 http://www.essepuntato.it/xml2earmark

5 Pattern Ontology: http://www.essepuntato.it/2008/12/pattern.

use of the elements still allowed by the schema, but trying to go for the simple core of the language as it is actually used by the majority of document authors. In particular we want to check if our theory based on eight simple structural patterns is able to capture and summarize the guidelines used by the authors of markup documents in their independent everyday practice.

How do real documents perform compared against our theory of patterns? In this section we discuss the experimental results of tests runs of our algorithm to determine the actual use of patterns by document authors. These tests largely confirm our hypotheses, but raise some unexpected issues.

In order to build a representative data set we collected about 1200 documents from eight different XML vocabularies. Vocabularies cover very different domains: from literary documents to technical documentation, from web pages to databases dumps, from conference proceedings to cookbooks. Documents vary a lot in terms of size and number of elements, and they were downloaded from very heterogeneous sources, all freely accessible on the Web. Table 3 briefly describes the sets of documents we studied, while full sources and the outcomes of our experiments are available at `http://www.essepuntato.it/2013/06/patterns/test`.

Table 3. The full dataset used to evaluate patterns.

	Vocabulary	**Source**	**#files**	**Min size (bytes)**	**Max size (bytes)**	**Avg size (bytes)**	**#elements**
1	GXL	GXLThe full set of examples in the official documentation of GXL 1.0, available at: http://www.gupro.de/GXL/	21	578	32635	4733	14
2	RecipeML	RecipeMLThe first five archives of recipes from the Squirrel's RecipeML Archive, available at: http://dsquirrel.tripod.com/recipeml/indexrecipes2.html	498	895	7196	2526	16
3	MusicXML	MusicXMLThe full MusicXML test-suite used to test the LilyPond program, available at: http://lilypond.org/doc/v2.17/input/regression/	127	828	41760	6587	273
4	FictionBook	FictionBookSome randomly downloaded books available at: http://fictionbook-lib.org/	100	69379	3464352	650375	61
5	EPrintXML	EPrintXMLSome randomly downloaded descriptors form the Caltech Institute public repository, available	50	3768	123081	26660	80

		at: http://caltechln.library.caltech.edu/eprints/					
6	XHTML	XHTMLThe full version of the Koran published by LiberLiber and freely available at: http://www.liberliber.it/	125	5551	261855	33328	31
7	DocBook	DocBookSome randomly downloaded papers from the proceedings of the Balisage Series Conferences, available at: http://www.balisage.net/	117	3283	161337	61053	64
8	TEI	TEI Some randomly downloaded files from the Gutemberg Project available at: http://www.gutenberg.org/	90	40245	1965027	445865	92

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

Results are encouraging. A summarized view is given in Figure 6, where each point corresponds to a set of documents.

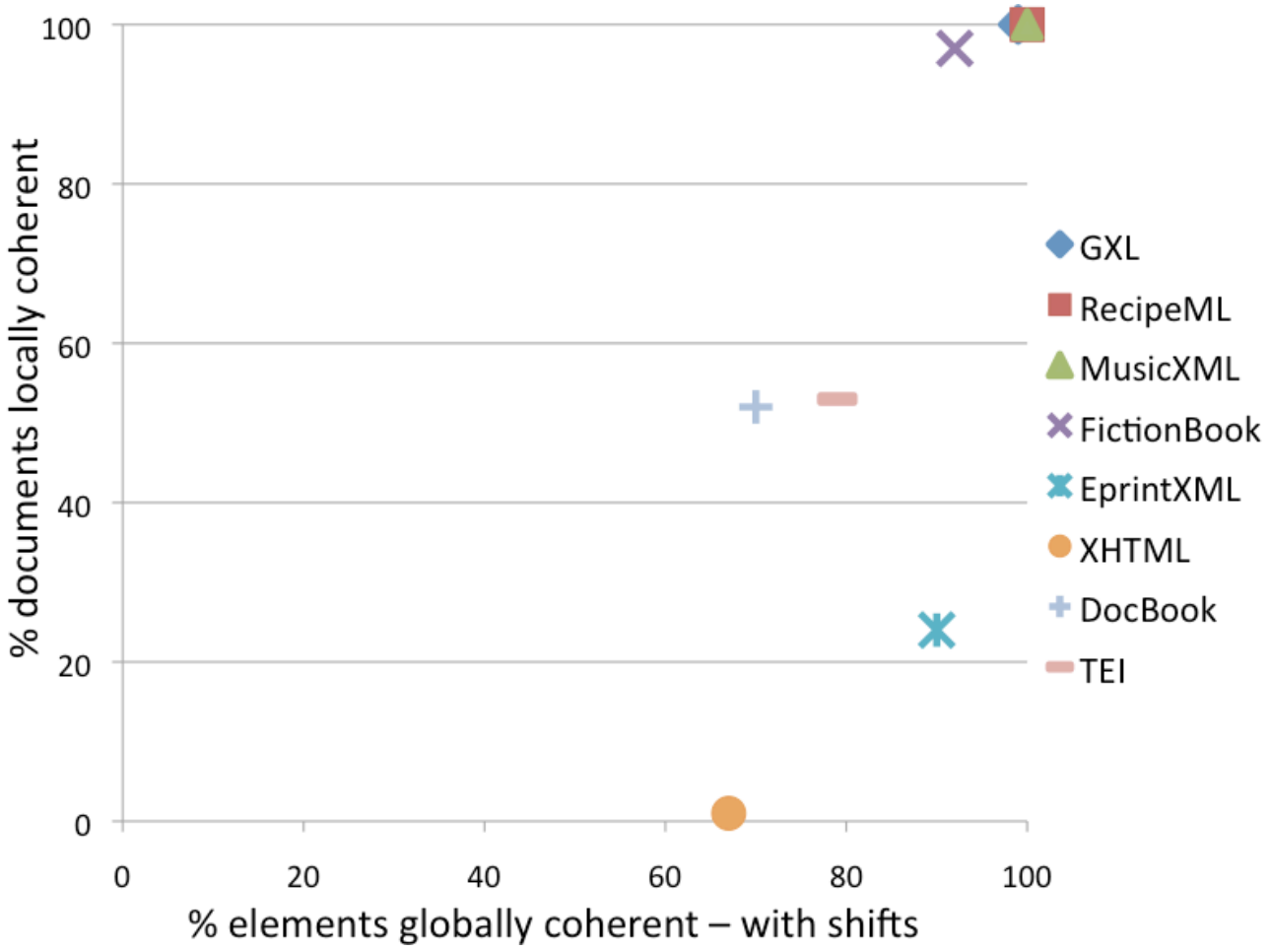

Figure 6. The percentage of locally coherent files and globally coherent elements for each language in the dataset.

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

Moreover, a detailed analysis of discrepancies shows that frequent and blatant misuses are always generated by a small number elements that impact on other elements. It is worth remarking that we did not analyse the definition of the elements in the vocabulary scheme (DTD or XML Schema or whatever), but rather we examined how these elements are actually used in real documents.

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

Full adherence or convergence to patterns

In three experiments we obtained global coherency. These vocabularies define the structure of graphs (GXL), musical scores (MusicXML) and ingredients of recipes (RecipeML). For each of them, in fact, it was possible to automatically derive a univocal patterns scheme by applying shifts. Table 4 summarizes these results.

Table 4. The result of checking patterns on three very structured vocabularies, which adhere to our theory natively or after a normalization phase.

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

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

- ***GXL***: 21 files using the Graph eXchanges Language, a format to describe graphs and define constructs such as edges, nodes and relations in a very structured way. Each file in the dataset is locally coherent. 11 elements out of 14 are used in exactly the same way in all files (80% of the total elements in the data set). By applying

global shifts and containers generalization, we easily managed to found a single pattern scheme valid for the whole dataset.

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

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

Large adherence

For two sets the number of elements that could not bring to local nor global coherency was low and we can conclude that they largely adhere to patterns. Table 5 summarises our results.

Table 5. The result of checking patterns on some vocabularies, which adhere largely to our theory.

	Set	**#files**	**# locally** ma direi **coherent**	**# elements**	**# elements generating local incoherency**	**# elements globally coherent (no global shifts and containers generalization)**	**# elements globally coherent (with global shifts and containers generalization)**

4	FictionBook	100	97 (97%)	61	1 (2%)	34 (56%)	56 (92%)
5	EPrintXML	50	12 (24%)	80	2 (3%)	57 (71%)	72 (90%)

- ***FictionBook***: 100 documents compliant with FictionBook, an XML vocabulary to encode the structure of e-books, using a total of 61 elements. We can conclude that a large part of the schema substantially uses patterns. Three documents were locally incoherent: in most cases, the problem was with the element `emph`; the authors used often this element to emphasize entire paragraphs. In some cases the element was placed outside of the paragraph, in others just inside, around the textual content of the paragraph, in others around pieces of text, in others around inline elements (`strong`, `sup`). Such differences made impossible a straight interpretation of the element. Other troublesome cases are `empty-line` (recognized as *milestone* in 81 files and as *field* in 1 file, since it contained one whitespace character) and `text-author` (recognized as *atom* in 36 files and as *table* in 1 file, since the content was structured in a sequence of very short paragraphs). For these 2 elements no shift or reduction was possible. We consider this an acceptable result.
- ***EprintXML*** The collection we studied was composed of 50 files encoding metadata about scientific papers, theses, reports and teaching material. The number of locally coherent files was very low: 12 over 50. This apparently rather bad result can be mitigated by observing that errors were connected to only two elements: `type` and `url`. In these files, in fact, these elements are recognized sometimes as *atoms* and sometimes as *fields,* thereby preventing any shift. Looking at data more carefully an interesting aspect comes to the light. Both these elements are direct children of the element `item` and are always used as *fields*. Everything would work if the element `item` was attributed to *record*. Unfortunately this element has been used in an odd way in a few bibliographic references, whose data was all specified as a plain text within one `item`, with one line for each entry and no internal structure. Such an odd choice made the algorithm recognize `item` as *block* and therefore `type` and `url` as *atoms*. The fact that there is no reachable coherency does not imply that the authors have preferred a different organization, but, in our mind, it is a side effect of the poor use of some elements. The other interesting point is that the errors on `type` and `url` impact only part of the dataset. In fact, we managed to assign one single pattern to 61 elements over 72 (90% of the total) by applying global shifts and containers generalization. The rest of the elements could not be restructured as patterns.

Partial adherence

The search of our patterns on some other vocabularies did not produce fully satisfactory results. That happened especially with languages that provide users several constructs and choices: the presence of content models that combine the same elements in very different ways, the nature of the languages that cover heterogeneous needs and narrower cases, the unconventional usage of some constructs make documents far from our pattern-based model. The results, discussed in detail in the following subsections, are summarized in Table 6.

Table 6. The result of checking patterns on some vocabularies, which adhere partially to our theory.

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

- ***XHTML***: We collected 125 pages linked to each other and corresponding to different parts of the same book. The number of locally incoherent files was very high: 124 out of 125. Such inconsistencies depend on just 2 elements, that basically generated the same problem in all these files: `table` and `a`. The overall layout is organized through nested tables: some cells contain only logos and extra information, others contain menus and navigational buttons (that are again organized in tables), others just text content. Moreover, each page contains a navigation menu expressed as a table containing `a` elements to go back and forward and to access the table of content. The use of the same t `table` and `a` elements for such a variety of purposes makes it impossible to assign them a single pattern and, as a consequence, overall results are distorted. Isolating these errors, we achieved good results. The elements can be split in three groups: 16 elements that were assigned the same pattern for all files (51%), 5 elements that can be reduced to a single pattern via global shifting and generalization (16%), and 10 elements (33%) that are problematic and confirm that the openness of the XHTML schema leads authors to create documents that are syntactically valid but, in our opinion, still unclear from a structural point of view.

- ***DocBook (Balisage)***: Our experiment was on a collection of 117 DocBook files, for a total of 64 different

elements. We found 55 locally incoherent documents (47% of the total). Differently from our previous experiments, several elements were involved in these errors. In fact, there were 15 elements (23% of 64) that generated local incoherency, although only 5 of them were incoherent in more than 10 files. We established that most of the 'errors' were the result of the multivalent use (allowed by the Balisage DTD) of particular elements. For instance, some authors used the element `figure` within a paragraph thus implicitly assigning it the *popup* role, while other times it has been used as direct child of containers and therefore recognised as yet another *container*. This variability should be interpreted neither as an error of the authors nor as a conflict between the pattern *popup* and *container*. Rather, it simply means that different authors used the same element in different ways. In particular, the element `figure` describes a precise structure according to its documentation, i.e. a block containing a display element (such as a `mediaobject`) and a title, which can be aligned to (typical of *containers*) or unaligned from (i.e. floating, typical of *popups*) the main flow of text. Global shifts and generalizations helped us to move towards coherency but this set is admittedly far from our pattern-based model. The 64% of the elements were actually given one single pattern even without reductions, while the final percentage was of 70%.

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

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

APPLYING PATTERNS TO VISUALIZATION AND

NAVIGATION: PVIEWER

The ability of patterns to capture the most relevant classes of structures and to express in a rigorous but simple way their relationships can be exploited to build novel tools for XML documents. Patterns allow us to build viewers that do not require users to directly master XML technologies but offer intuitive interfaces to read documents, moving within their components and extracting relevant information. The crucial aspect is that these tools are independent of the document vocabularies since they work directly on their pattern-based representation. Since patterns can be extracted through automatic processing, these applications work on *any document without any knowledge of its original schema*. Thus, they are very helpful whenever the vocabulary is unknown, not available or available in a different version.

We experimented this approach in *PViewer*. It is a proof-of-concept Java and XSLT implementation that takes as input an XML document and produces an HTML page, plus some CSS and Javascript, with its content and structures. Figures 7, 8 and 9 shows some zoom-in views of a possible output of PViewer generated automatically from a XML file randomly chosen in our dataset (and briefly described in the rest of the section).

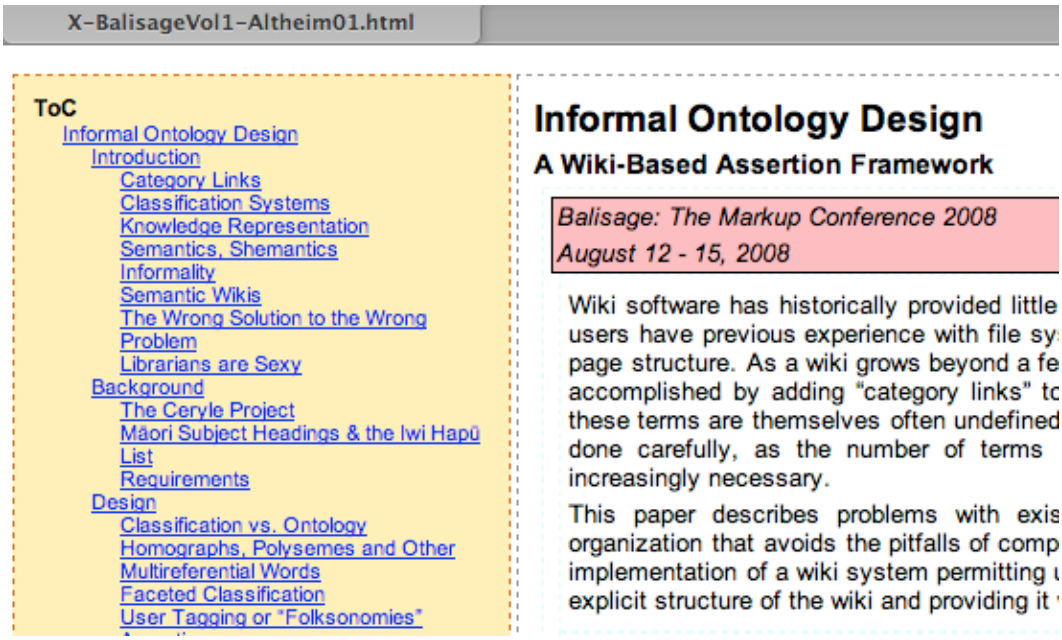

Figure 7. Basic visualization of a XML document in PViewer. The first blocks of the documents are shown in the right, sided by an automatically-generated table of content.

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

briefly described below.

Pattern identification. PViewer exploits the algorithm presented in Section "Determining structural patterns in documents" to identify patterns in any XML document. Two outputs are possible: in case of local coherency the algorithm produces one single map where each element is associated to one pattern; if not, it associates multiple patterns to some elements. In that case PViewer implements some reduction rules (basically, selecting the most general pattern within each sub-set) to produce a new map where each element is associated to only one pattern. This makes the overall approach work also on documents that are not locally coherent, with acceptable results.

Conversion and generation of presentation rules. PViewer translates the original XML file into a HTML page composed of generic containers, blocks and inlines, associated to some CSS rules. Elements and presentation rules are generated from the previous map, and convey the structural meaning of each pattern. For instance, as shown in Figure 8: containers are nested and added a border to clarify their containment relation, inline elements are highlighted with a darker background in contrast to plain text, milestones are substituted with images clicking on which users can read their XML source and attributes.

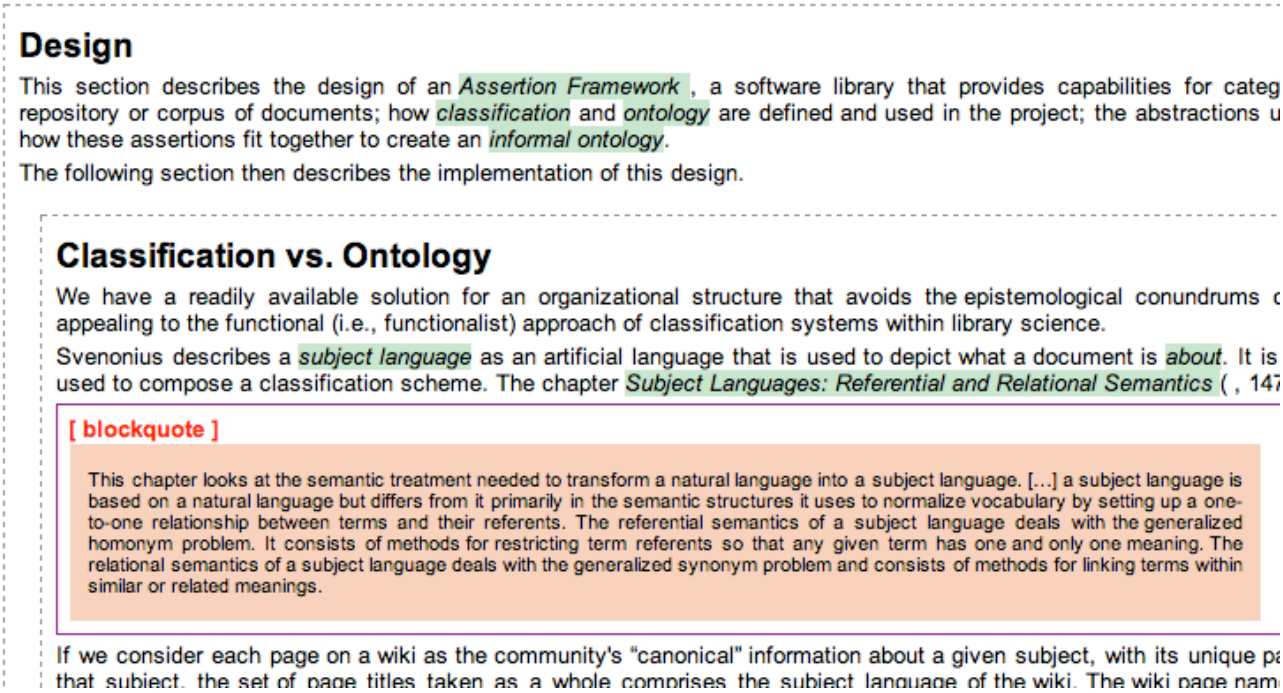

Figure 8. Details of visualization in PViewer: Inlines use a darker background, and Popups can be expanded on request. The hierarchical organization of Containers is highlighted through dashed borders.

Besides showing how nested containers appear in PViewer, this image shows how it handles Inlines and Popups: the

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

Information synthesis and extraction. Finally PViewer exploits the pattern-based classification of the elements in order to generate some extra pieces of the navigation interface. An example is the table of content shown in the left part of the screenshot in Figure 7, that gives users a clear insight of the overall structure of the document and its hierarchical components, and supports internal navigation. This table is created automatically from the data on HeadedContainers, as titles are mapped into labels of the index.

Patterns can also be exploited to extract a preliminary index of terms. In the patterns model, in fact, text fragments that carry a specific meaning within a flow of text are *Atoms* or *Inlines*. PViewer extracts all these fragments, removes some stop words and organizes them in alphabetical order. Figure 9 shows a zoom-in view of the PViewer index of terms. The terms under letters 'A' and 'B' are visible in the image. The whole index is shown to the user when clicking on the 'Terms' button on the left.

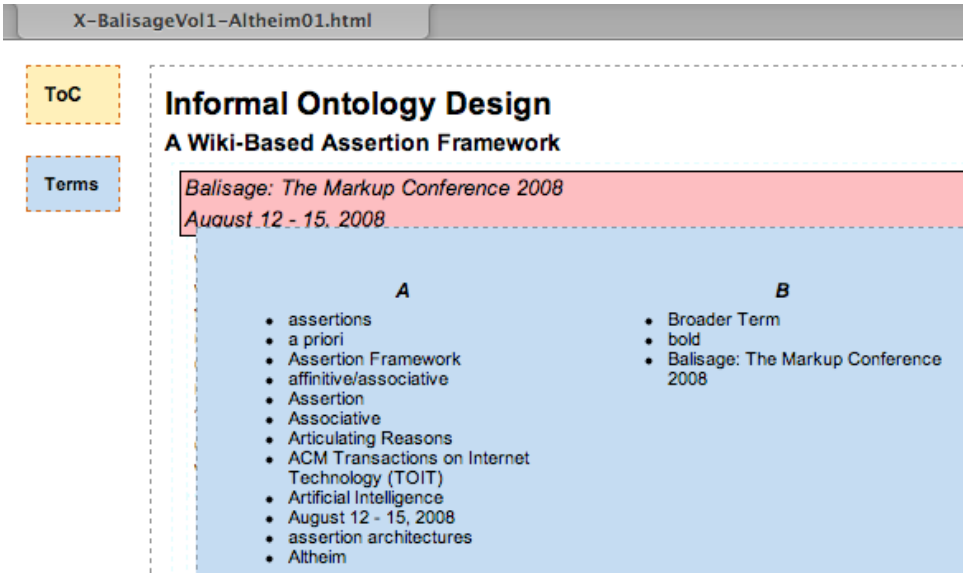

Figure 9. A zoom-in view of the basic index of terms generated by PViewer.

There are several improvements possible for this component. For instance, we plan to add support for counting the

number of occurrences of each term, filtering out some terms (for instance, by also integrating external linguistic components), linking terms to their occurrences in the text, aggregating statistical data, and so on.

The current PViewer implementation can be further developed in several directions. The current prototype, in fact, confirmed that patterns provide a rather good conceptualization for building vocabulary-independent applications. Results were positive since we managed to generate reasonable presentation for all documents even without specific knowledge of the language the documents were written in. Focusing on practical applications is the next step of our research: we plan to refine the current PViewer visualization, to add new features and to carry on exhaustive tests with the final users. These enhanced tools will be built on top of the theoretical model presented in this work.

CONCLUSIONS

One of the key points of this work is that the theory and the algorithm we propose are schema-independent: patterns can be exploited to make sense of documents regardless of the availability of schemas, tools and presentation stylesheets. The logic formulas behind our theory of patterns guarantee that no trivial configuration is eventually used neither at the local level, i.e., for the markup elements within a particular document, nor at the global level, for all the elements of a larger set of documents in the same XML vocabulary. For instance, it is impossible to assign the same pattern, e.g. *inline*, to the whole set of elements being examined, since this would result in making the ontological characterisation of patterns described in Section "A theory of patterns" totally inconsistent. In addition, the shift rules and the container generalisation mechanism we introduced (both in locally- and globally-defined scenarios) guarantee the identification of the most meaningful choice of patterns in terms of granularity: the algorithm retrieves always the most specific pattern for an element given its structure in terms of context and content model. This guarantees the largest set of patterns to be considered and, if possible, selected.

Making sense of documents by exploiting patterns in one of the next directions of our research. Our goal is to enable the description of the parts of a document at different levels of abstraction. For instance, we are working on models and tools to capture the structural, rhetorical and argumentative functions of the same text, in order to ease the document processing by humans and subsequent applications (Di Iorio, Peroni, Poggi, Shotton & Vitali, 2013). The fragmentation of documents into functional units has proved to have great potentialities in supporting access, navigation and comprehension of digital documents (Zhang, 2012).

Retrospective analysis and semi-automatic refactoring are further areas we plan to explore. We plan to investigate how

the authors/communities use specific constructs and validation rules. For example, we want to investigate whether there are differences in the way in which a given language is used. In particular, we want to compare the production of heterogeneous communities of authors (e.g., from different disciplines, with different backgrounds, etc.) that use the same language, in order to further verify the validity and the adequacy of our theory. In addition, we want also to investigate how the internal structure of documents, the formal definition of schemas, and the suggested community guidelines concerning the use of a particular schema can be improved by adopting patterns.

REFERENCES

- Colazzo, D., Sartiani, C., Albano, A., Manghi, P., Ghelli, G., Lini, L., Paoli, M. (2002). A typed text retrieval query language for XML documents. In Journal of the American Society for Information Science and Technology, 53 (6): 467-488. DOI: 10.1002/asi.10059.
- Dattolo, A., Di Iorio, A., Duca, S., Feliziani, A.A., Vitali, F. (2007). Structural patterns for descriptive documents. In Baresi, L., Fraternali, P., Houben, G. (Eds.), Proceedings of the 7th International Conference on Web Engineering 2007 (ICWE 2007). Berlin, Germany: Springer. DOI: 10.1007/978-3-540-73597-7_35
- DeRose, S., Durand, D. (1994). Making Hypermedia Work: A User's Guide to HyTime. Boston, Massachusetts, USA: Kluwer Academic Publishers. ISBN: 9780792394327
- Di Iorio, A., Gubellini, D., Vitali, F. (2005). Design patterns for document substructures. In Proceedings of the Extreme Markup Languages 2005. Rockville, MD, USA: Mulberry Technologies, Inc. http://conferences.idealliance.org/extreme/html/2005/Vitali01/EML2005Vitali01.html (last visited July 13, 2013).
- Di Iorio, A., Peroni, S., Poggi, F., Shotton, D., Vitali, F. (2013). Recognising document components in XML-based academic articles. To appear in Proceedings of the 2013 ACM symposium on Document Engineering. New York, New York: ACM.
- Di Iorio, A., Peroni, S., Poggi, F., Vitali, F. (2012). A first approach to the automatic recognition of structural patterns in XML documents. In Proceedings of the 2012 ACM symposium on Document Engineering: 85-94. New York, New York: ACM. DOI: 10.1145/2361354.2361374
- Di Iorio, A., Peroni, S., Vitali, F. (2011). A Semantic Web Approach To Everyday Overlapping Markup. In Journal of the American Society for Information Science and Technology, 62 (9): 1696-1716. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. DOI: 10.1002/asi.21591
- Di Iorio, A., Peroni, S., Vitali, F. (2011). Using Semantic Web technologies for analysis and validation of structural markup. In International Journal of Web Engineering and Technologies, 6 (4): 375-398. Olney,

Buckinghamshire, UK: Inderscience Publisher. DOI: 10.1504/IJWET.2011.043439.

- Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Boston, Massachusetts, USA: Addison-Wesley. ISBN: 0201633610.

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

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

- Horrocks, I., Patel-Schneider, P. F., McGuinness, D. L., Welty, C. A. (2007). OWL: A Description Logic Based Ontology Language for the Semantic Web. In Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F. (Eds.), The Description Logic Handbook: Theory, Implementation and Applications (2nd edition): 458-486. Cambridge, UK: Cambridge University Press. ISBN: 9780521876254.

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

- Krotzsch, M., Simancik, F., Horrocks, I. (2011). A Description Logic Primer. Ithaca, New York, New York: Cornell University Library. http://arxiv.org/pdf/1201.4089v1 (last visited July 13, 2013).

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

- Motik, B., Patel-Schneider, P. F., Parsia, B. (2012). OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation, 11 December 2012. World Wide Web Consortium. http://www.w3.org/TR/owl2-syntax/ (last visited July 13, 2013).

- Presutti, V., Gangemi, A. (2008). Content Ontology Design Patterns as practical building blocks for web

ontologies. In Li, Q., Spaccapietra, S., Yu, E. S. K., Olivé, A. (Eds.), Proceedings of the 27th International Conference on Conceptual Modeling (ER 2008). Berlin, Germany: Springer. DOI: 10.1007/978-3-540-87877-3_11

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

- Text Encoding Initiative Consortium (2005). TEI P5: Guidelines for Electronic Text Encoding and Interchange. Charlottesville, Virginia, USA: TEI Consortium. http://www.tei-c.org/Guidelines/P5 (last visited July 13, 2013).

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

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

- Yeh, J. F., Wu, C. H. and Chen, M. J. (2008). Ontology-based speech act identification in a bilingual dialog system using partial pattern trees. Journal of American Society for Information Science and Technology, 59 (5): 684-694. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. DOI: 10.1002/asi.20700

- Zhang, L. (2012). Grasping the structure of journal article: Utilizing the functions of information units. In Journal of American Society for Information Science and Technology, 63 (3): 469-480. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. DOI: 10.1002/asi.21680

- Zou, J., Le, D., Thoma, G. R. (2007). Structure and Content Analysis for HTML Medical Articles: A Hidden Markov Model Approach. In Proceedings of the 2007 ACM symposium on Document engineering (DocEng07). New York, New York: ACM. DOI: 10.1145/1284420.1284468.