Structural Patterns for Descriptive Documents

Antonina Dattolo[1], Angelo Di Iorio[2], Silvia Duca[2],
Antonio Angelo Feliziani[2], and Fabio Vitali[2]

[1] Department of Mathematics and Applications R. Caccioppoli,
University of Napoli Federico II, Italy
dattolo@unina.it
[2] Department of Computer Science, University of Bologna, Italy
{diiorio,ducas,afelizia,vitali}@cs.unibo.it

Abstract. Combining expressiveness and plainness in the design of web documents is a difficult task. Validation languages are very powerful and designers are tempted to over-design specifications. This paper discusses an offbeat approach: describing *any* structured content of *any* document by only using a very small set of patterns, regardless of the format and layout of that document. The paper sketches out a formal analysis of some patterns, based on grammars and language theory. The study has been performed on XML languages and DTDs and has a twofold goal: coding empirical patterns in a formal representation, and discussing their completeness.

Keywords: Patterns, grammars, descriptive schemas, completeness.

1 Introduction

The World Wide Web has become the greatest repository of information ever existed. The more the centrality of the WWW increases, the more web documents become heterogeneous and complex. We do not perceive *heterogeneity* as a problem, to be solved by flattening all those languages into a plain, unspecific and incomplete one. Neither we want to define '*The*' universal exhaustive language able to describe any domain and any application. What we are rather looking for is a general model to design languages and documents, in order to make them simple, complete and processable.

This work is part of a more ambitious project aiming at defining, separating and extracting constituents of *any* document so as to reformulate a few of them, to reuse some of them in different contexts, or to convert and mix subcomponents. Our overall approach relies on the definition of *abstract and generic formats* able to fully describe the most relevant bits of digital documents, regardless of their actual format, layout and storage. In particular, we consider *any* document as a composition of five components (dimensions) independent but connected each other: content, structure, presentation, behavior and metadata. We are not interested here in the details of our model (a deeper discussion can be found in [DDID07]) but we only refer to the *structure*. Our goal is discussing how *any* structured content can be *described* by a very small set of objects and composition rules. We already presented in [DIGV05] some *patterns*

L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS 4607, pp. 421–426, 2007.

to capture the most common structures of digital documents. What we want to do here is resuming that analysis and proposing a theoretical framework to discuss some properties of those patterns.

In particular, this paper sketches out a grammar-based proof of patterns' completeness. The interested reader is referred to [DDID07] for the complete proof. The point here is understanding what 'completeness' means in the context of descriptive documents, and how the language theory help us in proving it. Researchers have already used grammars for XML, in different contexts: [MLMK05] classified and compared schema languages, [Via01] discussed opposition and synergy between databases and XML, and [BMNS05] studied the relation between DTDs and XML Schema. Our goal is quite different: proving that *any* DTD can be transformed into a *descriptive* one, based on few patterns but able to capture the same structural information. Our analysis is performed over XML-based languages and DTDs, for sake of simplicity, but could be extended to other languages with appropriate modifications.

The paper is organized as follows: section 2 introduces a taxonomy of different levels of descriptiveness, section 3 presents our pattern-based model, section 4 defines grammars for (common or pattern-based) DTDs. Finally the completeness of the pattern-based language is sketched out.

2 Structures for Descriptive Documents

The first step of our work consists of understanding which constructs and rules are really needed for descriptive schemas. The distinction between prescriptive and descriptive languages have been widely studied in the literature [Ren00]. What these approaches change is the role itself of the validation [Pie01]. Traditionally validation is *strict*, because it is used as a "go/non-go" gauge to verify *in advance* whether or not a data set conforms to a set of requirements. A *loose* validation is rather used to capture abstract and structural information about a text. Both strict and loose validations are useful. What is important is designing languages and schemas by keeping in mind their features and differences, and applying them in right contexts.

Different *levels of descriptiveness* exist, depending on what designers reckon as important, what can be relaxed, what can be omitted, what can be expressed in a different way. To discuss these levels, we use DTD-based examples since they are shorter and more direct but we can use any other language, in exactly the same way. We identify six relevant levels of descriptiveness:

- **Prescriptive (PRE):** a prescriptive DTD imposes a set of rules which all matching documents must follow. Prevent errors in a production chain, based on strict validation.
- **Descriptive No Alternatives (DNA):** a descriptive DTD without alternatives do not allow users to force a choice between two (or more elements). The basic idea is that alternatives are meant to inhibit incorrect structures, but they are not required when all documents already exist and the DTD is used to describe all those documents.

- **Descriptive No Cardinality (DNC):** a descriptive DTD without alternatives can be further generalized by relaxing constraints over the cardinality of each single element. The idea is that by forcing cardinalities some documents could be considered invalid, even if they belong to the same class.
- **Descriptive No Order (DNO):** constraints over the order can be relaxed as well. Imposing order is something extremely useful when invalid documents obstruct a complex process, but it makes much less sense when the goal is identifying components. A descriptive document is not meant to say where each object is located, but which objects compose the document itself.
- **Super Descriptive (SD):** relaxing both constraints over cardinality and order, besides alternatives, designers can create abstract DTDs which consider any object as a sequence of repeatable and optional elements. These DTDs are meant to only define the set of objects of the documents.
- **(Un)Descriptive (UD):** relaxing any constraint designers could say that anything includes anything. Not useful in practice, those DTDs are only mentioned to complete our spectrum.

Table 1 shows a simple DTD declaration, accordingly to each model:

Table 1. Different levels of descriptiveness

Descriptiveness level	Content Model
PRE - Prescriptive	<!ELEMENT X (A, (B | C), D*)>
DNA - Descriptive No Alternativess	<!ELEMENTX (A, (B?, C?), D*)>
DNC - Descriptive No Cardinality	<!ELEMENT X (A*, (B*, C*), D*)>
DNO - Descriptive No Order	<!ELEMENT X (A & (B? & C?) & D*)>
SD - Super Descriptive	<!ELEMENT X (A | (B | C) | D)* >
UD - (Un)Descriptive	Any

On the basis of previous analysis [DIGV05], we identify the DNO paradigm as a good solution to describe documents' structures. Actually we did not cite directly DNO but we studied situations where such descriptiveness is enough to express everything users need. Moreover we proposed and discussed some patterns, concluding that by adopting these and only these patterns, all those descriptive situations could be covered.

3 Patterns for Descriptive Documents

The set of patterns we proposed in [DIGV05] is very small:

- **Marker:** an empty element, in case enriched with attributes, whose meaning primarily depends on its position within the context.
- **Atom:** a unit of unstructured information. An atom contains only plain text and is meant to indicate a specific role or semantics for that information
- **Block and Inline:** a block of text mixed with unordered and repeatable inline elements that, in turn, have the same content model. They are used to model any objects which ultimately carry the text written by the author.

- **Record:** a set of heterogeneous, unordered, optional and *non-repeatable* elements. Records are first used to group simple units of information in more complex structures, or to organize data in hierarchical subsets.
- **Container:** a set of heterogeneous, unordered, optional and *repeatable* elements. The name itself emphasizes the generality of this pattern, used to group diversified objects, repeated and collected together.
- **Table:** a sequence of homogeneous elements. Tables are used to group similar objects into the same structure and, also, to represent repeating tabular data.

A deeper discussion of each pattern is out of the scope of this paper (see our previous work for details) but some properties deserve some more space. First, patterns are *orthogonal*: each of them has a specific role and covers a specific situation, and no content model is repeated. Second, specific rules are imposed over the class of objects allowed in each content-model. For instance, an inline element can be contained only within a block, a container cannot directly contain plain text, a record or a table cannot be contained in a block, and so on.

Wrappers. Our approach relies on the methodical use of specific elements, called *wrappers*, which allow us to transform a generic DTD into a pattern-based one and to guarantee the *homogeneity* of content models. Every time a content model contains a mixed presence of repeated elements and single ones (or alternatives), a new (wrapper) element is created. It will substitute that 'wrong' declaration fragment, inheriting the content model. Consider for instance an element declaration of type `<!ELEMENT X (A,(B|C))>`; we don't want a sequence and a choice at the same time. Then we create a new element `W` (`<!ELEMENT W (B|C)>`) and, substituting it in the previous declaration, we obtain a homogeneous declaration `<!ELEMENT X (A,W)>`.

Moreover, the introduction of wrappers permits to "by-pass" all those situations where a constraint among patterns is violated. Consider for instance, a container element declared as `<!ELEMENT C (A|B)*>`, when `B` is an inline. A new block element, the wrapper `W` (`<!ELEMENT W (#PCDATA|B)*>`) can be created and the `C` definition can be changed in `<!ELEMENT C (A|W)*>`.

All changes introduced by wrappers are then targeted to "clean" (or homogenize) structures and to make documents more descriptive.

4 Formal Representation of Patterns

In this section we sketch out a formal analysis of the completeness of our patterns, based on language theory. Complete definitions and proofs can be found in [DDID07]. The basic idea consists of deriving properties of validation schemas by analyzing grammars which produce them, as proposed by [MLMK05].

We chose DTDs because they are more direct, but similar considerations could be extended to other languages like XML-Schemas[TDMM01] or RelaxNG[Mur00]. Although these languages are more powerful, in fact, creating and even reading the corresponding grammars would be much more difficult and

Table 2. Our pattern-based grammar P

```
[p01] elementdecl            ::= markerelementdecl | atomelementdecl | blockelementdecl
                                 | inlineelementdecl | recordelementdecl | containerelementdecl
                                 | tableelementdecl
[p02] markerelementdecl      ::= '<!ELEMENT' S MarkerName S markercontentspec S? '>'
[p03] atomelementdecl        ::= '<!ELEMENT' S AtomName S atomcontentspec S? '>'
[p04] blockelementdecl       ::= '<!ELEMENT' S BlockName S blockcontentspec S? '>'
[p05] inlineelementdecl      ::= '<!ELEMENT' S InlineName S inlinecontentspec S? '>'
[p06] recordelementdecl      ::= '<!ELEMENT' S RecordName S recordcontentspec S? '>'
[p07] containerelementdecl   ::= '<!ELEMENT' S ContainerName S containercontentspec S? '>'
[p08] tableelementdecl       ::= '<!ELEMENT' S TableName S tablecontentspec S? '>'

[p09] markercontentspec      ::= 'EMPTY'
[p10] atomcontentspec        ::= '(' S? '#PCDATA' S? ')'
[p11] blockcontentspec       ::= maicontentspec
[p12] inlinecontentspec      ::= maicontentspec
[p13] maicontentspec         ::= '(' S? '#PCDATA' (S? '|' S? maiName)+ S? ')*'
[p14] recordcontentspec      ::= '(' S? mabrctName '?'? (S? '&' S? mabrctName'?'?)* S? ')'
[p15] containercontentspec   ::= '(' S? mabrctName (S? '|' S? mabrctName)* ')*'
[p16] tablecontentspec       ::= '(' S? mabrctName S? ')*'

[p17] maiName                ::= MarkerName | AtomName | InlineName
[p18] mabrctName             ::= MarkerName | AtomName | BlockName
                                 | RecordName| ContainerName | TableName
```

time-consuming. More important, the vast majority of existing schemas proved to be structurally equivalent to DTDs [BMNS05].

We first define a grammar P (shown in Table 2) able to generate all the pattern-based DTDs. Productions [p01-p08] are used to declare the seven different patterns, while the remaining ones are introduced to specify their content models.

Note that we perform some simplifications to make simpler and clearer the analysis: we (i) omit attributes declarations, (ii) do not consider some unusual declarations as `(#PCDATA)*` (equivalent to`(#PCDATA)`), (iii) do not consider the terminal symbol '+' both for shortness and because it could be associated to the terminal '*' from a descriptive perspective. Note also that we introduce the terminal symbol '&', that in SGML syntax means that all elements must occur in any order, in order to better formalize the DNO model.

We then compare our grammar with a general grammar G, provided by the W3C [BPSMM00], that produces all the possible DTDs.

The result is that for each DTD, producible from G, it exists a pattern-based DTD, producible from P, which is equally descriptive at DNO level. To do it, we present a reduction algorithm, which applied to a DTD, generates a pattern-based DTD, equally descriptive at DNO level. Formally:

Proposition 1. *Given L(P) and L(G), let r: L(G) → L(P) be a function that implements our reduction algorithm; we want to state that*

$$\forall d \in L(G)\ \exists p \in L(P)\ \ni' d \xrightarrow{r} p \tag{1}$$

with p and d equally descriptive at DNO level.

The symbol $\xrightarrow{r}$ indicates that d is reduced to p applying the function r.

The proof is a case-by-case analysis of the production rules of G, and, when needed, a reduction operation (based on wrappers insertion) to transform those rules into descriptive ones. During this reduction process, we relax some constraints, prescribed in grammar G; in this way, the set of documents, accepted by the pattern-based DTDs, generated by P, is at least large as the set of documents generated by the original DTD.

5 Conclusions

In this paper we discussed how any document *structure* can be described by a very small set of objects and composition rules. Moving off an analysis of different levels of descriptiveness, we presented a grammar-based formalization of our patterns and some sketches of a formal proof of their completeness (whose extended version can be found in [DDID07]). In the future, we plan to inspect, in the same formal way, other properties like correctness and minimality.

References

[BMNS05] Bex, G.J., Martens, W., Neven, F., Schwentick, T.: Expressiveness of xsds: from practice to theory, there and back again. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, pp. 712–721. ACM Press, New York (2005)

[BPSMM00] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: Extensible Markup Language (XML) 1.0 (2000), `http://www.w3.org/TR/REC-xml`

[DDID07] Dattolo, A., Di Iorio, A., Duca, S., Feliziani, A.A., Vitali, F.: Patterns for descriptive documents: a formal analysis (2007), `ftp://ftp.cs.unibo.it/pub/techreports/2007/2007-13.pdf`

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

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

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

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

[Ren00] Renear, A.: The Descriptive/Procedural Distinction is Flawed. Markup Languages: Theory and Practice 2(4), 411–420 (2000)

[TDMM01] Thompson, H.S., Beech D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures (2001), `http://www.w3.org/TR/xmlschema-1/`

[Via01] Vianu, V.: A web odyssey: from codd to xml. In: PODS '01. Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–15. ACM Press, New York, NY, USA (2001)