Extreme Markup Languages 2007®

Montréal, Québec
August 7-10, 2007

Converting into pattern-based schemas: a formal approach

Antonina Dattolo
Department of Mathematics and Applications R. Caccioppoli, University of Napoli Federico II

Angelo Di Iorio
Department of Computer Science, University of Bologna

Silvia Duca
Department of Computer Science, University of Bologna

Antonio Angelo Feliziani
Department of Computer Science, University of Bologna

Fabio Vitali
Department of Computer Science, University of Bologna

Abstract

A traditional distinction among markup languages is how descriptive or prescriptive they are. We identify six levels along the descriptive/prescriptive spectrum. Schemas at a specific level of descriptiveness that we call "Descriptive No Order" (DNO) specify a list of allowable elements, their number and requiredness, but do not impose any order upon them. We have defined a pattern-based model based on a set of named patterns, each of which is an object and its composition rule (content model), enough to write descriptive schemas for arbitrary documents. We show that any schema can be converted into a pattern-based one without loss of information at the DNO level (*invariant conversion*). We present a formal analysis of invariant conversions of arbitrary schemas as a demonstration of the correctness and completeness of our pattern model. Although all examples are given in DTD syntax, the results should apply equally to XSD, Relax NG, or other schema languages.

Converting into pattern-based schemas: a formal approach

Table of Contents

Converting into pattern-based schemas: a formal approach

Antonina Dattolo, Angelo Di Iorio, Silvia Duca, Antonio Angelo Feliziani, and Fabio Vitali

§ Introduction

Given any set of XML document, there is an open number of schemas that validate all members of the set, and reject all non-members. Not only schema languages allow some linguistic variety in expressing the same constraints, but also different constraints can be devised that actually accept and reject the same instances.

Of course, when dealing with XML documents, validation is only part of the story. Classification (i.e., the ability to associate meaning and procedures to each part of the document) is also a key aspect for XML applications, and many would say actually more important than validation. When classifying documents, the correct association of each element with its label is more important than clearly specifying the set of rejected documents. Thus not only different schemas can be associated to the same set of documents for validation purposes, but also the different emphasis on validation vs. classification can imply or require different features to be made available in the schemas themselves, further increase their total number.

Given the different emphasis that can be placed in operating with XML documents, schemas to deal with the same set of documents can be written with different perspectives in order to look for different things. Design patterns can be applied in the design of such schemas that can alleviate the problem of determining the important and the not-so-important things to check in an XML document.

In particular, in the past we have fruitfully applied design patterns to descriptive schemas, whenever it is more important to describe what is in a document than to check for violation to structural rules. This paper has grown out of a question raised at Extreme 2005, after the presentation of our work on a pattern-based approach for descriptive schemas [DIGV05]. On that occasion, we presented a very small set of patterns meant to design any descriptive schema for arbitrary documents. During the question-time, we were recommended to further investigate the properties of such model in a formal way.

The goal of this work is then to demonstrate the completeness of those patterns, as a means for *descriptive validation* (validation against descriptive schemas) of document classes. We want to show that any schema can be automatically converted into an 'equivalent' one (w.r.t. descriptiveness), which is exclusively based on our patterns, and that such schema can accept the same class.

This of course requires both explaining what concept of "equivalence" we have adopted and which is the actual meaning of such conversion. Actually we first need to clarify what we mean for 'descriptive schemas' and 'descriptive validation' and what we mean these descriptive schemas for. In many ways, this work starts off the traditional dichotomy between prescriptive and descriptive markup languages [Ren00] [Qui96]. We note that many prescriptive constructs are used in descriptive contexts (and viceversa), and that these approaches are very often mixed with each other. This may happen for different reasons: the fact that some schemas are widely known and users prefer to adapt and reuse them rather than re-building new ones (still, not a bad choice considering that a lot of benefits derive from such choice), as well as the fact that designers receive only partial information about the domain they are modeling and the exact kind of validation that will be concretely needed. Last but not least, it is undeniable that constraining choices available in validation languages are often too powerful, and easily lead us to over-design of schemas.

In 2005 we started discussing 'descriptive schemas' and proposed some substantial simplification which can be made by considering the nature and purposes of documents being analyzed. These simplifications allow us to write simple and well-engineered schemas which better fit descriptive needs. This work further delineates and formalizes those initial ideas.

To this end, we need to refine our notion of descriptiveness. A first result is the identification of some subclasses of descriptive approaches for markup languages (that we call here "levels of descriptiveness"). Each subclass corresponds to some common needs and preferences according to which designers can relax constraints, express information by adopting specific rules and patterns, and reformulate declarations. For instance, we indicate with DNA (Descriptive No Alternatives) those schemas where alternative choices are so irrelevant as to be omittable, with DNC (Descriptive No Cardinality) those schemas where rules over cardinality can be relaxed, and with UD (Undescriptive) those schemas that accept all content models.

In particular, we identified the DNO level (Descriptive No Order) as a good solution for the 'descriptiveness' objectives we discussed in our previous paper: such schemas do not use alternatives, express cardinality of each element and relax constraints over their order. Moving off such classification, and after resuming the discussion about expressivity and scope of DNO schemas, we will present a reduction algorithm to generate a DNO schema from any schema that is invariant with respect to the descriptiveness of the constraints.

The analysis is divided in two main parts: a formal study of the conversion algorithm of XML DTDs, based on grammars and language theory, and an informal discussion about XML-Schema and RelaxNG schemas. Future version of this work will be applying formal techniques to these languages as well. The paper is then organized as follows: section 2 discusses descriptive schemas and constraints relaxation on that class of documents, section 3 refines our notion of descriptiveness and invariant conversion, sections 4 and 5 formalize grammars for XML DTDs and present our reduction algorithm; finally section 6 analyzes XML-Schema and RelaxNG in order to extend our approach to other languages too.

§ Descriptive markup languages and patterns

Simplifying descriptive schemas

The analysis of descriptive markup languages is not new. Many researchers interrogate about the inherent nature and purposes of these languages, by comparing them with prescriptive approaches. For instance, [Qui96] focused on DTDs: a prescriptive DTD may be designed to create new material or to mark up existing material, and prescribes a set of rules which all matching documents must follow; a descriptive DTD is used to create an electronic version of material that already exists (of course, a descriptive model may also be used to create new documents) and describes structures that exist, rather than forcing any particular structure.

A very good perspective to study these approaches and figure out application areas for each of them is investigating the concept of validation they differently implement. As outlined by Piez [Pie01] two classes of validation can be identified, roughly corresponding to prescriptive and descriptive schemas: *strict* and *loose*. The traditional way of conceiving validation is ”strict”, because validation is used as a ”go/non-go” gauge to verify in advance whether or not a data set conforms to a set of requirements. The example provided by Piez explains very well the role of such a validation: the publishing process can be likened to an assembly line and validation is a control phase that prevents errors and makes the whole system work. When a document fails validation, there is something wrong with it, something that has to be changed in the document itself. Strict validation is useful (and sometimes necessary) as a means to split a complex job into sub-activities, that can be accomplished by different actors with different skills and facilities.

Even if less frequent, an opposite perspective is alike interesting: using validation to describe documents and to capture *a posteriori* structural information about a text. Piez defined such a process as a *loose* validation. It might be important to trace those features of the text important to the author or to encoder, rather than those constraints essential for subsequent operations over that text. Moreover it may happen that some features of documents are still undefined when designing schemas, as well as some instances (which should be considered valid) are still unknown. Thus, a descriptive schema is not something that exists before an instance, as a set of rules to be followed; in a sense, it derives from instances, as an *ex post facto* expression of what can be discovered from them. As a consequence, such a schema is not composed by fine-grained declarations that capture variations and exceptions, but it is composed by generic rules that capture the overall meaning of a set of documents.

A clear identification of contexts of use of schemas has (or, at least, should have) a great impact over their design. In particular, we have noticed how descriptive schemas can be simplified by carefully taking into account their real scope and objectives. Two examples, we also brought in 2005, help us in explaining such simplification process (check our previous paper for some more examples and a deeper analysis):

Alternatives

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

Figure 1: Expressing alternatives in a DTD-like syntax

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

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

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

A corresponding descriptive rule would therefore be:

Figure 2: Expressing alternatives with a descriptive rule

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

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

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

Mixed content models

Mixed content models are by definition used when describing semi-structured text flows that are part of larger contexts. Paragraphs that have meaningful subparts inside are natural candidates for mixed content models.

Each individual sub-element of a paragraph specifies some special meaning or style on the wrapped text. For this reason, all text elements within a sub-element of a paragraph are also part of the paragraph. In this view, it makes little sense that sub-elements of a mixed content paragraph is allowed to contain data that is not part of the paragraph text flow, since this could be difficult to identify without precise advance knowledge the meaning of the sub-element itself and its further subparts.

Thus the only reasonable forms of mixed content models should be:

Figure 3: Defining a mixed-content model to model paragraphs

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

or, at most, if we want to exclude further nesting inside sub-elements,

Figure 4: Defining a mixed-content model to model paragraphs, excluding further nesting inside sub-elements

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

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

Patterns for descriptive document structures

The main topic of the paper we presented at Extreme 2005 was a set of patterns that are enough for writing any descriptive schemas. We provided there a quite exhaustive description of our patterns by focusing on their features, applications and mutual relationships. We do not want to repeat here that analysis, but we need to focus on some features relevant for conversion (and formal proof) we explain later. Table 1 shows our patterns, defined by a very small set of objects and composition rules:

Table 1: Patterns

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

The first thing to notice is the introduction of a new pattern, called **container**, used to model all those circumstances where diversified objects are repeated and collected together. The name emphasizes the genericity of this pattern: a container, in fact, is an unordered set of repeatable and heterogeneous elements:

Figure 5: The container pattern

```
<!ENTITY % Blocks "( para | table | li )" > <!ELEMENT body
           %Blocks; > <!ELEMENT section %Blocks; > ...
```

As expected, the content-model of a container includes markers, atoms, blocks, records, tables and containers themselves. Only raw text and inlines are excluded, because they should be wrapped within a block. Containers are related to both records and tables of our model. On the one hand, they share the classes of elements included in a record (blocks, inlines, records, etc.): what changes is only the repeatability of those elements, but the order is not relevant in both cases. On the other hand, they are related to tables for their repeatability. The only difference is that items of a container are heterogeneous, while those within a table are homogeneous. We use two separated patterns just to emphasize the difference between homogeneous and heterogeneous structures. Patterns in fact are meant to clearly distinguish the structural role of the objects.

Actually we are discussing about the specialization of the pattern **container** into a **hierarchical container**, which distinguished the title from its actual content (unordered set of repeatable elements). However, this difference is not relevant here: what is important, is the presence of an object able to gather and collect heterogeneous elements under the same logical wrapper.

Wrappers play a very important role in our model. They are specific elements used to gather uniform information, to group elements, and to emphasize relationships among them. Basically they spread the information over the depth of the document in order to decrease the need for complex constructs, and make explicit mutual connection among elements. For instance, every time a content model contains a mixed presence of repeated elements and single ones (or alternatives), a new (wrapper) element can be created to better model that scenario. It will substitute that 'wrong' declaration fragment, inheriting the content model. Consider for instance an element declaration `<!ELEMENT X (A,(B|C))>`. A new element W (`<!ELEMENT W (B|C)>`) can be introduced and substituted in previous declaration, in order to obtain a homogeneous declaration `<!ELEMENT X (A,W)>`. In turn, the declaration (`<!ELEMENT W (B|C)>`) can be changed into (`<!ELEMENT W (B?, C?)>`), when alternatives are actually irrelevant (i.e., descriptive scenarios). Note that the new definition does not impact the expressivity of the schema: in particular, it does not change its descriptiveness since it correctly generalizes the meaning and structure of the valid documents.

Another point is worth being remarked about patterns: specific rules are imposed over the class of objects allowed in the content-model of each of them. 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. Table 2 shows these constraints (each row indicates elements allowed in the content-model of each pattern).

Table 2: Composition rules over patterns

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

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

Wrappers can be use 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) documents structures.

§ Invariant conversion among descriptive schemas

Refining the notion of descriptiveness

The choice between descriptive and prescriptive models primarily depends on the relation between the process of actual writing a document and the process of encoding it. Descriptive models reflect the fact that authors have worked before the creation of the schema. Then, designers are in charge of accommodating variations, exceptions and generalize the features of a (possibly large) set of documents. Still, such generalization is not a pre-defined and univocal process. Designers in fact may want to relax some constraints, omit or reformulate some definitions, express information in a different way according to different criteria and preferences. What we presented in the previous section is only one of the possible solutions towards descriptiveness.

A deeper analysis leads us to identify 6 different subclasses of descriptive approaches, which we summarize in the following paragraphs. We call them **levels of descriptiveness**. Note also that our definition, although refer to DTDs, can be applied to any other validation language. What matter are the objectives and principles behind each paradigm:

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 (including variations and exceptions otherwise unpredicted by a strict/prescriptive DTD).

Descriptive no cardinality (DNC) a descriptive DTD without alternative 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. Such validation is not meant to prevent errors, but to describe existing resources.

Descriptive No Order (DNO) constraints over the order can be relaxed as well. Imposing an order is something extremely useful when invalid documents obstruct a complex process, but it makes much less sense when the goal is identifying subcomponents. A descriptive document is not meant to say where each object is located (a presentation layer can change that property), but which are the objects contained in the document itself.

Super Descriptive (SD) relaxing both constraints over cardinality and order, besides alternatives, designers can create abstract DTDa which consider any object as a sequence of repeatable and optional elements (as in the example). Apparently vague, 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 3 shows a very simple DTD declaration, transformed according to all these models.

Table 3: Descriptiveness levels example

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

Introducing the notion of Invariant Conversion

Our previous analysis, based on the paper we presented in 2005 [DIGV05], implicitly identified the DNO paradigm (with some important variations) as a good solution to design generic schemas for document structures. Although we did not explicitly mention DNO, in fact, we studied situations where such descriptiveness (relaxing alternatives and order, but maintaining cardinality) is enough to express everything users need. In the same paper, we proposed some patterns and concluded that most of those descriptive situations can be modelled by adopting these and only these patterns. The goal of this work is proving that such pattern-based approach is complete according to a given notion of equivalence.

Our notion of equivalence is strictly related to the idea of 'levels of descriptiveness'. In particular, we exploit the strong separation between competencies and requirements implemented at each level. As stated before, in fact, choosing a level of descriptiveness depends on the context of use. Each level implies a set of constraints which can be relaxed, a set of properties which are interesting to be maintained, and a set of simplifications which can be made without loosing expressivity.

Another point is very important about these levels: their partial sorting. The same table we used to present them reflects that feature: levels embody increasing grades of generalization or, in reverse order, decreasing levels of prescriptiveness. The more constraints and impositions are relaxed, the more a higher level of the hierarchy is involved. Thus, starting from the lowest PRE level (where concrete and specific rules impose strong constraints over documents), our classification goes up to DNA (which relaxes constraints over alternatives) which, in turn, goes up to DNC and DNO. In a sense, they can be considered at the same 'meta-level' since they both relax constraints on a single dimension by keeping unchanged the other one. By merging these approaches, we obtain schemas at SD level, up to (Un)Descriptive (UD) one, which accepts any document.

On the basis of such considerations, the term *fully equivalent at a given level* indicates that the information carried by two schemas is exactly the same *under the constraints implied by that level of descriptiveness*. To excess, all schemas can be considered equivalent at the higher level (UD) since no constraint actually exists. Similarly, at DNO level no matter whether or not a schema expresses alternatives, as well as at DNC level it is not relevant to count the number of the elements. All these constraints are relevant at lower levels (down to the PRE level, where no rules can be relaxed) but can be neglected when dealing with schemas from a higher perspective.

We then define an **invariant schema conversion to a given level D** as a "**transformation from a whichever input schema A into a specific output schema B, which respects constraints and rules imposed by D level and preserves D descriptiveness of schema A in schema B**".

An example of invariant conversion at DNO level is shown below:

```
<!ELEMENT bibliography (info, (title, subtitle?)?, (mixedcontent |
entries))) >
```

```
<!ELEMENT bibliography (info & title? & subtitle? & mixedcontent? &
entries?) >
```

Note that an invariant conversion makes sense only when associated to a specific level of descriptiveness. A converted schema does not express exactly the same information of the original one, rather the same information relevant for that context. Consider for instance the example in table 3: the schema at DNO level cannot be used to prevent the co-presence of the B and C elements (as the prescriptive one) but give readers an exhaustive description of the elements contained in X, if their position and co-existence are not relevant.

The above mentioned discussion about motivations and use-cases for using DNO schemas can now be applied to DNO invariant conversion. By converting schemas into DNO equivalent ones, we obtain instances which better fit descriptive scenarios. Our starting claim (about the completeness of our patterns according to a notion of equivalence) can be now reformulated: we want to prove that **for any schema there exists an invariant conversion at DNO level which transforms it into a schema exclusively based on our patterns**. We want to state that any schema can be generalized (and normalized) into a new one which 'descriptively validates' the same set of documents and only uses our patterns: the point is that all simplifications and reductions on the input schema do not impact constraints and rules imposed at DNO level. From now on, we will use the term 'invariant conversion' to indicate such class of reductions, implying they hold at DNO level.

We applied invariant conversions to some elements' declarations of well-known markup languages, in order to produce their pattern-based 'relatives'. Once again, the point is not raising objections on those original declarations, but showing how they are simplified in a DNO context. Consider for instance the glosslist element of DocBook (version 4.5) whose content-model is shown in the first line of fig. 6:

Figure 6: The glosslist declaration in DocBook 4.5 and its pattern-based reformulation

```
<!ELEMENT glosslist (blockinfo?,
          (title,titleabbrev?)?, glossentry+)> <!ELEMENT glosslist (blockinfo?,
          title?, titleabbrev?, glossentry+)> <!ELEMENT glosslist (blockinfo?,
          title?, titleabbrev?, glossentries?)> <!ELEMENT glossentries
          (glossentry+)>
```

The operator '?' after the sequence title and titleabbrev, which is in turn optional, sounds quite unnatural: why do designers need to repeat it twice on the titleabbrev element? The reason is that a titleabbrev makes no sense without a title, and designers wanted to prevent the existence of an isolated `titleabbrev`. Although legitimate in a prescriptive environment, such a declaration can be substituted by a more general one in a descriptive scenario. The second declaration in fig. 6 validates also a glossary with only an abbreviated title but captures the same information at DNO level, that is the existence of title and abbreviated title for a glossary.

The presence of a sequence (of elements `glossentry`) mixed with non-repeated elements (`bookinfo` and titles) also conflicts with our pattern-based vision. However, it is evident that the `glossentry` elements are logically contained in a wrapper, even if that wrapper were not explicitly used in the document itself. The introduction of a virtual element, say `glossentries`, makes explicit and processable the uniformity among `glossentry`s and distinguishes them from the rest of the `glosslist` content-model. By paying some extra verbosity, the resulting schema clearly highlights the overall structure of a list of entries in a glossary.

The case of `glosslist` is not isolated. Similar considerations can be repeated about any DTD (or better, any schema written in any language), whenever declarations are designed for prescriptive purposes and generalized into descriptive ones. A similar example can also be found in the Extreme Markup Conference DTD, by looking at the element `deflist`, which encodes a two-column list of definitions. The corresponding declaration, shown in fig. 7, can be transformed into a pattern-based one by adding a virtual wrapper for the `item` elements (which plays the same role of the previous `glossentries`) and by relaxing constraints over the co-presence of a term name and explanation. Although an explanation without a term makes a little sense, and should be prevented when creating new documents, the (second and descriptive) declaration in fig. 7, which considers an heading as a record of both optional elements `term.heading` and `def.heading`, equally expresses the basic information of a definition list.

Figure 7: The `deflist` declaration in the Extreme Markup DTD and its pattern-based reformulation

```
<!ELEMENT deflist (title?, (term.heading,
            def.heading?)?, def.item+)> <!ELEMENT deflist (title?,term.heading?,
            def.heading?, def.items?)> <!ELEMENT def.items (def.item+)>
```

Prescriptive constraints can also be relaxed on the TEILite Specifications, keeping invariant declarations at DNO level. Consider the element `publicationStmt`, which groups information about the publication and distribution of an encoded text. Its content model, shown in fig. 8, is meant to define two different forms of statement, which cannot be mixed each other. A `publicationStmt` is a choice between two sequences of some common information (captured by the `%m.Incl;` entity) preceded by paragraphs or other publication-related elements. The complex structure of this content-model derives from the need of validating homogeneous sequences, which exclusively start with paragraphs *or* different elements. A descriptive declaration does not need to prevent such co-existence but aims at indicating which elements appear and how many times. By also transforming alternatives into records of optional elements, according to the theory discussed in our previous paper, this declaration could even be converted in the very general schema shown in fig. 8. It could be further improved by introducing wrappers, which make explicit the relationship among uniform and repeatable elements.

Figure 8: The `publicationStmt` declaration in TEILite and its pattern-based reformulation

```
<!ELEMENT publicationStmt ( ( p, (%m.Incl;)*)+ | (
           (publisher | distributor | authority | pubPlace | address| idno | availability | date ),
           (%m.Incl;)*)+ )>

<!ELEMENT publicationStmt ( ( p?, publisher?,
            distributor?, authority?, pubPlace?, address?,idno?, availability?, date?),
            (%m.Incl;)*)+
```

§ Invariant conversion towards pattern-based documents: a formal analysis

Grammars and formal languages

In order to deeply analyze patterns we performed a formal analysis, based on language theory. In formal language theory, a language is defined as a set of words built over a set of terminal symbols, and grammars define rules to combine together terminals through productions. Our idea is to derive properties of validation languages (whether pattern-based or not) by analyzing the grammars which produce these languages.

We will present a formal study of invariant conversions for DTDs. We will show that, for any DTD, it exists an invariant conversion (once again, at DNO level) which transforms it into a pattern-based one.

We chose DTDs because they are simpler, easier to be read, and more direct. The reverse of the medal is the fact that DTDs are less expressive than other schema languages, and many schemas (for instance, all those written in XML-Schema [TBMM01] or RelaxNG [CM01]) seem to be cut off from our discussion.

In this connection, we will later discuss differences between DTDs, XML-Schema and RelaxNG in terms of descriptiveness and invariant conversion. We will conclude that advanced features of XML-Schema and RelaxNG do not impact the descriptiveness of a schema, and invariant conversion is valid for those languages as well. A proof on DTDs is then meant to prove the completeness and soundness of a general conversion approach, independent from actual schema languages.

An empirical and quite surprising result is also interesting on this point: [BMNS05] discovered that the 85% of the existing XML-Schemas are equivalent to DTDs. Although they took random schemas which are probably not representative for all schemas currently used (one of the reason, for instance, is the fact that many tools automatically generate fairly elaborate schemas) they pointed out a very interesting trend, i.e. an undeniable strength of DTDs in the everyday life. Even narrowing the analysis on the only DTDs, would be a partial but interesting result.

Our proof consists of three main phases: (i) introducing a grammar which produces all the possible DTDs, (ii) introducing a grammar which produce all the DTDs based on our patterns (described in our previous work [DIGV05]) and (iii) presenting a reduction algorithm which applies an invariant conversion on each production of the grammars. In the following subsections, we introduce the general grammar G for DTDs and we propose our grammar P able to produce all the DTDs which use only our patterns, postponing the comparison between related languages to section 5.

The general grammar G

The general grammar G, provided by the W3C [BPSMM00], produces all the possible DTDs. To make it easier, we extracted some rules and worked only on them. In particular, we are interested in the element type declarations (summarized in figure 9) since they define the overall structure of a document.

Figure 9: General grammar G

```
[45] elementdecl ::= '<!ELEMENT' S Name S
           contentspec S? '>' [46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children [51]
           Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' | '(' S? '#PCDATA' S? ')' [47]
           children ::= ( choice | seq ) ( '?' | '*' | '+')? [48] cp ::= ( Name | choice | seq ) (
           '?' | '*' | '+')? [49] choice ::= '(' S? cp ( S? '|' S? cp )+ S? ')' [50] seq ::= '(' S?
           cp ( S? ',' S? cp )* S? ')'
```

Our grammar P

The grammar P aims at expressing in a formal way constraints and composition rules over pattern-based documents. The production rules of that grammar are summarized in figure 10. Productions [p01-p08] are used to declare the seven different patterns, while the remaining ones are introduced to specify their content models.

Figure 10: 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?)+ S? ')*' [p14] recordcontentspec ::= '(' S? mabrctName '?'? ( S?
          &' S? mabrctName '?'? S? )* S? ')' [p15] containercontentspec ::= '(' S?
          mabrctName ( S? '|' S? mabrctName S? )* S? ')*' [p16] tablecontentspec ::= '(' S?
          mabrctName S? ')*' [p17] maiName ::= MarkerName | AtomName | InlineName [p18] mabrctName
          ::= MarkerName | AtomName | BlockName | RecordName | ContainerName | TableName
```

We perform some initial simplifications to make simpler and clearer the analysis: for instance, we omit attributes declarations, and we do not consider some unusual declarations as `(#PCDATA)*` (that can be substituted with the equivalent `(#PCDATA)`). Moreover, we do not consider the terminal symbol '+' both for shortness and both because it could be associated to the terminal '`*`' from a descriptive perspective.

Another point is worth being explained: 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.

§ Invariant conversion between DTDs

Our proof is concluded by a point-to-point comparison between the languages generated by the two grammars, G e P. The goal is showing the existence of an invariant conversion at DNO level, for any DTD generated by the grammar G into a DTD generated from P. This property can be formally split in two sub-propositions:

Proposition 1 *Let L(P) and L(G) be the languages generated respectively by our grammar P and the general grammar G. L(P) is the set of all possible pattern based DTDs, while L(G) is the set of all possible DTDs.*
$L(P) \subset L(G)$

Proposition 2 *Let L(G) and L(P) be as above.*
$\forall\, d \in L(G) \exists\, p \in L(P)$

| for some reduction algorithm r: L(G) → L(P), $d \xrightarrow{r} p$ *and p and d are equally descriptive at DNO level.*

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

Proof. We want to demonstrate the existence, for any DTD, producible from G, of a pattern-based DTD, producible from P, which is equally descriptive at DNO level. To do it, we propose a constructive demonstration: we present a reduction algorithm, which applied to a DTD, generates a pattern-based DTD, equally descriptive at DNO level.

The algorithm for the transformation of a generic DTD in a pattern-based DTD consists of two phases, called *element types detection* and *refinement*: the first phase shows how each element X, generated by G, can be mapped in an element X' generated by P, and, in this way, associated at one of the seven possible patterns. The demonstration analyzes, in exhaustive way, the derivations generated by productions [46-51] of grammar G and matches them with equivalent (for derivations) productions of grammar P [01-18]; the second phase performs a cross check on identified patterns in order to assure that all pattern constraints (see Table 2) are respected; any violation is corrected applying a set of four reduction rules.

Element types detection. Now we apply an exhaustive analysis, deriving the language generated by grammar G and identifying the correspondence with productions and language generated by grammar P:

by applying productions [45] and successively [46], if we derive in G contentspec ⇒ 'EMPTY', equivalently we apply productions [p01], [p02] and [p09] in P and we derive markercontentspec ⇒ 'EMPTY'. In this case the element X is associated to a **Marker**.

By applying productions [45], [46] and [51], if we derive contentspec $\Rightarrow$ Mixed $\Rightarrow$ (#PCDATA), equivalently we apply productions [p01], [p03] and [p10] in P and we derive atomcontentspec $\Rightarrow$ (#PCDATA). In this case the element X is associated to an **Atom**.

By applying productions [45], [46] and [51], if we derive contentspec $\Rightarrow$ Mixed $\Rightarrow$ (#PCDATA | Name | ... | Name)*, then let n = Name | ... | Name, or in more general form n = N_1 | ... | N_m, with $m \geq 0$. If n = Ø, then contentspec $\Rightarrow_*$ (#PCDATA)* . Relaxing constraints, we reduce (#PCDATA)* $\xrightarrow{r}$ (#PCDATA) (case treated before). If n $\neq$ Ø, then contentspec $\Rightarrow$ Mixed $\Rightarrow$ (#PCDATA | N_1 | ... | N_m)* . In this situation, if $\exists$ i $\in$ {1, ..., m} $\ni$ ' X = N_i (that is, X includes it self in its content model), X is associated to an **Inline**; while if n $\neq$ Ø and $\nexists$ i $\ni$' X = N_i, with i = 1, ..., m, X is associated to a **Block**. In the first case, we apply in grammar P productions p[01], [p05], [p12], [p13] and [p17], while in the second case productions [p01], [p04], [p11], [p13] and [p17]. Since productions [11], [12] and [17] require that $\forall$ i, i=1, ..., m, N_i $\in$ MAI = {MarkerName, AtomName, InlineName}, if this constraint is not respected, in the refinement phase appropriate reduction rules must be applied.

By applying productions [45], [46], [47] and [49], if we derive contentspec $\Rightarrow$ children $\Rightarrow$ choice $\Rightarrow$ (cp | cp | ... | cp), we relax constraints and we associated to X a **Record**, reducing (cp | cp | ... | cp) $\xrightarrow{r}$ (cp? & cp? & ... & cp?). In this case, the equivalent productions in grammar P are [p01], [p06], [p14] and [p18]. The constraint is $\forall$ cp, cp $\in$ MABRCT = {MarkerName, AtomName, BlockName, RecordName, ContainerName, TableName}. As before, if this constraint is violated, appropriate reduction rules must be applied in the refinement phase.

By applying productions [45], [46], [47] and [49], if we derive: contentspec $\Rightarrow$ children $\Rightarrow$ choice* $\Rightarrow$ (cp | cp | ... | cp)* X is associated to a **Container** (applying equivalent productions [p01], [p07], [p15] and [p18]. In this case, if the constraint $\forall$ cp, cp $\in$ MABRCT = {MarkerName, AtomName, BlockName, RecordName, ContainerName, TableName}, is violated, in refinement phase, appropriate reduction rules must be applied.

By applying productions [45], [46], [47] and [49], if we derive contentspec $\Rightarrow$ children $\Rightarrow$ choice+, then we relax constraints, reducing choice+ $\xrightarrow{r}$ choice* (case treated before); if we derive contentspec $\Rightarrow$ children $\Rightarrow$ choice?, we relax constraints, reducing choice? $\xrightarrow{r}$ (cp? & cp? & ... & cp?) (case treated before).

By applying productions [45], [46], [47] and [50], if we derive: contentspec $\Rightarrow$ children $\Rightarrow$ seq $\Rightarrow$ (cp, cp, ..., cp), then we relax constraints, reducing (cp, cp, ..., cp) to (cp & cp & ... & cp). In this situation X is associated to a **Record** (applying equivalent productions [p01], [p06], [p14] and p[18]).

By applying productions [45], [46], [47] and [50], if we derive contentspec $\Rightarrow$ children $\Rightarrow$ seq* $\Rightarrow$ (cp, cp, ..., cp)*, equivalently we apply productions [p01], [p08], [p16] and [p18] in grammar P and X is associated to a **Table** and a record wrapper is introduced for cp, ..., cp.

Finally, by applying productions [46], [47] and [50], if we derive contentspec $\Rightarrow$ children $\Rightarrow$ seq? $\Rightarrow$ (cp, ..., cp)?, we relax constraints, reducing (cp, ..., cp)? $\xrightarrow{r}$ (cp? & ... & cp?) (case treated before); while, if we derive contentspec $\Rightarrow$ children $\Rightarrow$ seq+ $\Rightarrow$ (cp, ..., cp)+, then we relax constraints, reducing (cp, ..., cp)+ $\xrightarrow{r}$ (cp, ..., cp)* (case treated before).

In previous analysis, we have we have left out the derivation, obtained by applying productions [45] and [46]: contentspec $\Rightarrow$ 'ANY'; the reason is that, in practice, this approach is rarely used because it allows too much freedom, and therefore undermines the benefits that derive from defining document structures. Only for the sake of completeness, we mention it and we associate to it a Container wrapper, that can contain, directly or indirectly (by means other wrappers), any other element.

We note that, during this phase, the presence of 'cp' claims, in many situations, the introduction of appropriate wrappers.

The result of this phase is the reduction of each element to one (and only one) of the seven patterns, or, in alternative, is the introduction of a wrapper, which assumes the type of a given pattern. As consequence of this consideration, in order to complete our reduction, we need of checking that all elements respect inclusion constraints synthesized in Table 2. This is performed by next refinement phase.

Refinement. The second phase performs a cross check between elements to assure that all constraints are observed. Table 4 marks with an X only permitted combinations in elements' declarations, while marks with (x), x = 0, ..., 4, the situations in which a reduction rule must be applied.

Table 4: Composition rules over patterns and reductions

	EMPTY	Text	Marker	Atom	Block	Inline	Record	Container	Table
Marker	X	(0)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
Atom	(0)	X	(0)	(0)	(0)	(0)	(0)	(0)	(0)
Block	(0)	X	X	X	(1)	X	(3)	(3)	(3)
Inline	(0)	X	X	X	(1)	X	(4)	(4)	(4)
Record	(0)	(0)	X	X	X	(2)	X	X	X
Container	(0)	(0)	X	X	X	(2)	X	X	X
Table	(0)	(0)	X	X	X	(2)	X	X	X

For each element declaration one must checked that all contained element in the content model don't violate the constraints expressed in Table 2. Every time an element content model brakes a rule (i.e. an inline declaration contains a block in its content model) a specified reduction rule has to be applied according to the previous table (in the previous example the (1) reduction rule has to be applied). Some times a reduction rule can change the type classification of an element implying a complete recheck of all element declarations, making this algorithm iterative. We note that, during 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. In the following subsections each reduction rule will be specified.

Reduction rule (0) All cases marked with this reduction rule will never happen because the definition of the element excludes those cases. For example if an element has been recognized as *Marker* it is impossible that the declaration of the element contains other elements than the keyword 'EMPTY'.

Reduction rule (1) In the case that a block element is found in the declaration of other blocks or inline then a two-steps reduction has to be applied:

- the block element that is found inside other element will be no longer considered as a block but as an inline;
- having changed the type classification of an element all the element declarations have to be re-checked with the new type classification.

Reduction rule (2) In the case that an inline element appears inside a record, a container or a table element one must created a new block element that will substitute the inline element in all the wrong positions; the content model of the new block element will have to contain only the inline element. In this case, a re-check is not needed.

Reduction rule (3) In the case that a record, a container or a table element appear in content model of a block element, a three steps reduction has to be applied:

- remove the `offending-element` from the `block-element` containing it;
- create a new container-wrapper with content model of type `(block-element | offending-element)*`;
- substitute everywhere the `block-element` with the new container-wrapper.

In this case it's not needed a re-check.

Reduction rule (4) In the case that a record, a container or a table element appear in content model of an inline element than a three steps reduction has to be applied:

- remove the `offending-element(s)` from the `inline-element` containing it;
- push the removed `offending-element(s)` in all the parent block elements that contain as son or descendant the `inline-element`;
- having modified the content model of some elements, all the element declarations have to be re-checked.

This concludes the demonstration. *CVD.*

§ Extending the notion of invariant conversion: XML-Schema and RelaxNG

DTDs are widely used among designers because of their simplicity and plainness. Surprisingly, although more powerful validation languages exist, most of the existing schemas are still equivalent to DTDs [BMNS05]. Then, an invariant conversion among DTDs is a partial but very interesting result of our research.

The next step will be extending our approach to XML-Schema and RelaxNG. We plan to analyze them by exploiting language theory as well. A point is very important: our goal is not evaluating their expressiveness (many authoritative analysis exist, based on formal [MLM00] or informal [LDWCW00] [JR01] approaches) but their "descriptiveness". In particular, we want to prove that the reduction algorithm proposed in the previous section can be alike applied to any schema written in XML-Schema and RelaxNG.

We then need to verify how much each new feature of these languages impacts the descriptiveness. Note that XML-Schema and RelaxNG provide users much powerful functionalities, but most of them are not directly related to the matter of this paper, i.e. the (structured) content models of descriptive schemas. Then, the analysis can be focused only on some relevant aspects.

In [LDWCW00] Lee and Chu presented a comparative analysis of six schema languages, including DTDs and XML-Schema. We investigated each dimension proposed by the authors, in relation to descriptiveness and our patterns, and extended their analysis to RelaxNG. This section sketches our results and discusses how both XML-Schema and RelaxNG do not extend the "descriptiveness" of DTDs. As a consequence, our pattern-based approach can be extended to these languages as well.

Lee and Chu classified validation languages' features in 7 categories:

Schema

The features related to the overall organization, modularization and syntax do not impact the descriptiveness, since they do not cover definitions of content-models. The fact that XML-Schema and RelaxNG documents are written in XML, the fact that they are namespace-aware and the fact that they provide powerful mechanisms to include/import external schemas does not extend the set of choices and constructs already available in DTDs content-models.

Datatype

XML datatype can be classified in two types: simple and complex. Simple types (the only discussed in the "datatype" section of the original survey) do not impact structured and mixed content models, but allow designers to better define the text content of elements and attributes. Built-in types, user-defined types, modifiable facets give designers a more powerful control over atomic objects. In a descriptive scenario however they can all be generalized as strings (`(#PCDATA)`).

Note that RelaxNG provides a more flexible solution than XML-Schema, since it allows users to import and reuse any existing datatype set (and natively supports XML-Schema types). Even there customized and extensible datatype sets do not impact structured content models.

Attribute

XML-Schema and RelaxNG empower the management/validation of attributes. In particular, both of them allow users to associate simple types to an attribute, and overcome CDATA-related limitations of the DTDs. Moreover, RelaxNG makes possible choices among attributes, some co-constraints over their content, and so on. Although the relation between content-models and attributes is very interesting (traditional discussion about attribute/element trade offs can be found in [CR02]), attributes are temporarily out of the scope of this research. They in fact do not directly change structured and mixed content models.

Element

As expected, elements play a central role in our analysis. Lee and Chu identified seven aspects of the elements' management. Some of them are irrelevant for our discussion since they do not improve content-models expressiveness, like the presence of default values. Others were not discussed because supported by all the languages, like the presence of ordered sequence and choices among elements, or by none of them, like the open model (implemented for instance in Schematron [Jel05]). Other dimensions are indeed very important and will be discussed below:

1. **content-model**: all the schema languages we are analyzing support the same set of content-models: empty, text, element, or mixed. The mixed content model is particularly interesting since changes from DTDs to XML-Schema and RelaxNG (DTDs have only a type of mixed content model (`(#PCDATA | A | B)*`), while Schema and RelaxNG give users more control over the position and mutual relations among in-lines). In a descriptive environment, however, designers do not need to express constraints and rules over the position of the in-line elements. Any specific mixed content-model can be then transformed into a general and unordered declaration. In RelaxNG things are simpler, since the text is a `<rng:text>` element, which can be used as any other element. Even there any prescriptive declaration can be generalized in a XML DTD-like mixed content model.
2. **unordered-sequence**: the SGML & operator has been removed in XML-DTDs, for the sake of validation disambiguity. XML-Schema and RelaxNG restored that operator introducing `<xs:all>` and `<rng:interleave>`. Since our pattern-based model directly uses the unordered sequences (in the records), these operators do not raise issues. Actually, a remark is worth for RelaxNG: the operator `<rng:interleave>` has a more powerful interleaving semantics. Consider for instance `A & B*`. Expressed in RelaxNG, it matches any interleaving of a sequence containing a single A element and a sequence containing zero or more B elements; it thus allows the A element to occur anywhere, including between two B elements. Being more general and validating a larger set of documents, such a declaration is in line with our descriptive intent.
3. **min/max occurrences**: Both XML-Schema and RelaxNG allow designers to precisely define the minimum and maximum number of occurrences of any element. Such control, particularly useful in a prescriptive environment, is not needed in a descriptive one where users are interesting in *which* elements appear, more than *where and how many times*. Similar declarations can be then transformed into DTD-like declarations using the '`?`' and '`*`' operators.

Inheritance

XML-Schema gives great importance to simple and complex types, and provides users many features from object-oriented programming. In particular, types can be derived by restriction and extension. On the contrary, RelaxNG does not provide explicit support for type derivation. Since simple datatyping is not relevant here, the derivation of simple types is not interesting as well. Some remarks are indeed useful about derivation over complex types. XML-Schema allows designers to extend complex types by adding elements and attributes only at the end of a content model: any extended type can be then handled as a sequence of elements or as any other content-model enriched with attributes. Both these situations have already been handled in our analysis and pattern-based approach. Derivation for restriction, on the contrary, does not impact descriptiveness since users can design loose definitions and ignore prescriptive restrictions.

Being Unique Key

The simple ID/IDREF model of DTDs has been improved by XML-Schema and RelaxNG. While XML-Schema specifications directly provide powerful mechanisms to guarantee uniqueness of IDs, elements and attributes, RelaxNG moves issues related to the keys' uniqueness in a different standard (called RelaxNG Compatibility [CJMM01]). Identifiers do not impact content models, but the text content of specific attributes. As a consequence, they can be neglected in our analysis of LLC conversion.

Miscellaneous

The residual class proposed by Lee and Chu addressed other aspects which do not impact descriptiveness and invariant conversions. First of all, they also discussed some features not relevant because they are unsupported either by XML-Schema (and DTDs) or by RelaxNG, like the dynamic constraints (which can be turned off and on, upon specific conditions) or elements/attributes versioning (which allows authors to define different values of the same object in different versions of the same schema). Second, they included aspects related to description and documentation, like the possibility of adding annotations and documentation, as well as embedding HTML code or producing self-describing specifications. All these features are variably supported by XML-Schema and RelaxNG but do not change the expressiveness of the elements' declarations.

All these considerations are summarized in table 5, derived from the analysis of Lee and Chu. The table shows a fine-grained analysis of the features of DTDs, XML-Schema and RelaxNG and focused on their relation with descriptiveness and our pattern-based normalization. Future revisions of this work will provide much more details about each feature, and a formal analysis of its impact on invariant conversion.

Table 5: DTDs, XML-Schema and RelaxNG features analysis

Features	**DTD 1.0**	**XML Schema 1.0**	**RelaxNG**	**LLC XML-Schema**	**LLC RelaxNG**
			Schema		
syntax in xml	No	Yes	Yes	*norelevance* Syntactical differences do not change expressiveness.	
namespace	No	Yes	Yes	*norelevance* Namespaces do not impact content-models	
include	No	Yes	Yes	*norelevance* Inclusions and imports do not impact content-models	
import	No	Yes	Yes	*norelevance* Inclusions and imports do not impact content-models	
			Datatype		
built-in type	10	37	any	*relevance* Simple types do not impact structured content-models, and can be reduced into a CDATA in a descriptive environment.	Valid for RelaxNG as well
user-defined type	No	Yes	Yes		
domain constraint	No	Yes	Yes		
null	No	Yes	Yes		
extensibility	No	No	Yes		RELAX NG is not tied to a single set of datatypes. However new simple types do not change the structured content models.
			Attribute		
default value	Yes	Yes	Yes	*noanalysis* Our analysis does not cover attributes and	Valid for RelaxNG as well
choice	No	No	Yes		Choice between attributes is an interesting extra-

				their relations with content models.	feature of RelaxNG, not relevant here.
optional vs. required	Yes	Yes	Yes		Valid for RelaxNG as well
domain constraint	Partial	Yes	Yes		
conditional definition	No	No	Partial		
				Element	
default value	No	Partial	Partial	*norelevance* Default values allow designers to pre-define types and content-models, but do not change their expressiveness.	
content model	Yes	Yes	Yes	*relevance* Both schema languages support empty, text, element, or mixed content models. DTDs have only a type of mixed content model (`(#PCDATA | A | B)*`), while XML-Schema provides a more powerful mechanism. In a descriptive environment, designers do not need to express constraints and rules over the position of the in-line elements. Any XML-Schema mixed content-model can be then transformed into a general DTD-like block or in-line.	Valid for RelaxNG as well. In RelaxNG `#PCDATA` is a `<rng:text>` element and can appear in any position of a mixed content-model. The XML DTD-like declaration "generalizes" all these scenarios.
ordered sequence	Yes	Yes	Yes	*nodifference*	
unordered sequence	No	Yes	Yes	*relevance* Patterns use the SGML `&` operator, equivalent to the XML-Schema `<xs:all>`. XML-Schema unordered sequences are then directly included in our model.	In RELAX NG, the corresponding operator (`<rng:interleave>`) has a more powerful interleaving semantics. RelaxNG declarations are more general and validate a larger set of documents. No problem in terms of descriptiveness.
choice	Yes	Yes	Yes	*nodifference* Alternatives are less interesting and useful in a descriptive scenario.	
min & max occurrence	Partial	Yes	Yes	*relevance* In a descriptive environment authors do not need to precisely specify how many times an element can appear within another one. XML-Schema declarations about occurrences can be then transformed into DTD-like declarations using the '?' and '*' operators.	Valid for RelaxNG as well.

open model	No	No	No	*nodifference*	
conditional definition	No	No	Partial	*nodifference*	*relevance* Co-constraints importance in a descriptive context is very low. More than preventing erroneous instances of documents, in fact, designers aims at describing the general structure of a large set of documents.
<u>Inheritance</u>					
simple type by extension	No	No	Yes	*relevance* Simple types do not impact structured content-models, and can be reduced into a CDATA in a descriptive environment.	
simple type by restriction	No	Yes	Yes		
complex type by extension	No	Yes	No	*relevance* Derived complex type adds elements at the end of a content-model and can be then handling as a stand-alone complex type.	*nodifference*
complex type by restriction	No	Yes	No	*relevance* Restriction is not relevant in a descriptive environment, where designers can use general and loose definitions	*nodifference*
<u>Being unique or key</u>					
uniqueness for attribute	Yes	Yes	---	*norelevance* Uniqueness of IDs, elements and attributes do not impact contet-models and documents' structures	Moreover RelaxNG decouples identity-constraints and specifications of document structures.
uniqueness for non-attribute	No	Yes	---		
key for attribute	No	Yes	---		
key for non-attribute	No	Yes	---		
foreign key for attribute	Partial	Yes	---		
foreign key for non-attribute	No	Yes	---		
<u>Miscellaneous</u>					
dynamic constraint	No	No	No	*nodifference*	
version	No	No	No	*nodifference*	
documentation	No	Yes	Yes	*norelevance* Extra information about declarations, elements and attributes do not impact content-models	
embedded HTML	No	Yes	Yes	*norelevance* HTML embedding allows designers to describe and comment their schemas, but does not modify content models.	
self-describability	No	Partial	No	*norelevance* Self-describability is useful for implementers, but does impact the expressiveness of content-models	

Legend:

norelevance = does not change content-model expressiveness

relevance = changes content-model expressiveness

nodifference = supported (or unsupported) by both of the languages

noanalysis = not yet analyzed in detail

§ Conclusions

This paper is rooted in the traditional conflict between prescriptive and descriptive markup languages. Our first goal was to further investigate descriptive schemas and identify some subclasses of that approach, in order to prove properties of a pattern-based model we presented at Extreme 2005 and we actually use in publishing applications.

Although the paper has intentionally been focused on formal languages/schemas properties, in fact, we have also been working on a variety of tools which use internal formats based on our patterns. IsaWiki [DIV05], for instance, is on open publishing environment aiming at simplyfying web editing processes and allowing users to customize (the content of) any web page; IsaLearning [DFMSV06] is a chain of authoring tools which allows users to easily create high-quality learning-objects from raw input files.

The philosophy behind those applications is that a radical simplification of markup practice can facilitate creation of simple but expressive schemas and documents. In those contexts where data formats are meant to express information extracted by *a posteriori* analyses, in fact, the schemas for these formats could not be restrictive and express *a priori* rules of validation. On the contrary, users and applications take advantages from more general (descriptive) schemas which guarantee compatibility and 'tie' only a partial *but actually relevant* information. This work aims at consolidating our theory, by showing how to formalize and automate such a generalization/reduction process.

To go back to our original subject, we plan to further work on formal properties of our model. Besides extending our grammar-based analysis to other validation languages, we also plan to investigate minimality and correctness of our patterns. From an implementation perspective, instead, we will work on actual converters able to automatically transform schemas, as well as editors for pattern-based documents.

Acknowledgements

We thank the components of our reseach team, Michael Sperberg-McQueen and the anonymous reviewers for their helpful comments.

Bibliography

[BMNS05] G. J. Bex, W. Martens, F. Neven, and T. Schwentick, "Expressiveness of xsds: from practice to theory, there and back again" *In WWW ``05: Proceedings of the 14th international conference on World Wide Web*, New York, NY, USA, ACM Press, 2005.

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

[CJMM01] James Clark, and Makoto Murata, "RELAX NG DTD Compatibility", oasis-open.org, **http://www.oasis-open.org/committees/relax-ng/compatibility.html**, 03 Dec. 2001.

[CM01] James Clark, and Murata Makoto, "Relax NG", **http://relaxng.org/spec-20011203.html**, 03 Dec 2001.

[CR02] Robin Cover, "SGML/XML: Using Elements and Attributes", **http://xml.coverpages.org/elementsAndAttrs.html**, 26 Aug. 2002.

[DFMSV06] A. Di Iorio, A.A. Feliziani, S. Mirri, P. Salomoni and F. Vitali "Automatically Generating Accessible Learning Objects", *Journal on Educational Technology & Society*, Volume 9, Issue 4, 2006.

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

[DIV05] A. Di Iorio, Vitali, "From the Writable Web to the Global Editability", *Proceedings of ACM Hypertext '05*, ACM Press, Salzburg, Austria, 2005, pp. 35-45.

[Jel05] Rick Jelliffe, "Schematron 1.5", **http://xml.ascc.net/schematron/**.

[JR01] Rick Jellife, "The W3C XML Schema Specification in Context", O'REILL xml.com, **http://www.xml.com/pub/a/2001/01/10/schemasincontext.html**, 10 Jan. 2001.

[LDWCW00] Dongwon Lee, and W. Chu Whesley, "Comparative analysis of six XML schema languages", *SIGMOD Rec., ACM Press*, **http://doi.acm.org/10.1145/362084.362140**, New York, USA, 2000.

[MLM00] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. Extreme Markup Languages, 2000.

[Pie01] W. Piez, "Beyond the ``descriptive vs. procedural`` distinction", *The Extreme Markup Conference*, Montreal, Canada, 2001.

[Qui96] L. Quin, "Suggestive Markup: Explicit Relationships in Descriptive and Prescriptive DTDs", *The SGML 96 Conference*, Boston, MA, USA, 1996.

[Ren00] A. Renear, "The Descriptive/Procedural Distinction is Flawed", *Markup Languages: Theory and Practice*, 2000.

[TBMM01] Henry S. Thompson, David Beech, Murray Maloney, and Noah. Mendelsohn, *XML Schema Part 1: Structures*, **http://www.w3.org/TR/xmlschema-1/**, May 2001.

The Authors

Antonina Dattolo
Department of Mathematics and Applications R. Caccioppoli, University of Napoli Federico II
Napoli
Italy
dattolo@unina.it

Antonina Dattolo is an research associate at the Department of Mathematics and Applications "R. Caccioppoli" at the University of Naples Federico II. He holds a Laurea degree in Computer Science from the University of Salerno and a Ph.D. in Applied Mathematics and Computer Science from the University of Naples Federico II. Her research interests include markup languages; concurrent architectures for distributed hypermedia models; and . software agents. She is the author of several papers on distributed hypermedia models.

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

Angelo Di Iorio holds a Laurea degree and a PhD in Computer Science from the University of Bologna. His research interests include content management systems, web technologies, markup languages and digital publishing.

Silvia Duca
Department of Computer Science, University of Bologna
Bologna
Italy
ducas@cs.unibo.it

Silvia Duca holds a Laurea degree in Computer Science from the University of Bologna. Her research interests include web-semantic, web technologies, ontologies and markup languages.

Antonio Angelo Feliziani
Department of Computer Science, University of Bologna
Bologna
Italy
afelizia@cs.unibo.it

Antonio Angelo Feliziani holds a Laurea degree in Computer Science from the University of Bologna. His research interests include document management systems, web technologies, markup languages and e-Learning.

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

Fabio Vitali is an associate professor at the Department of Computer Science at the University of Bologna. He holds a Laurea degree in Mathematics and a Ph.D. in Computer and Law, both from the University of Bologna. His research interests include markup languages; distributed, coordinated systems; and the World Wide Web. He is the author of several papers on hypertext functionalities, the World Wide Web, and XML.

Extreme Markup Languages 2007®
Montréal, Québec, August 7-10, 2007
This paper was formatted from XML source via XSL
by Mulberry Technologies, Inc.