Beyond DTDs: constraining data content

José Carlos Ramalho Pedro Rangel Henriques

19 May 1998

Abstract

In most specific SGML applications, DTDs aren't enough to express what the designer had in mind. SGML is very good for structure specification. But, sometimes this structure is very loose, gives too much freedom to the user creating a margin for errors. The solution is constraining that structure according to the final document type purposes.

This way the user (who writes the documents according to that DTD) will not have full control of his data; he will be enforced to obey certain domain range limitations or certain information relationships.

In this paper we will discuss the way we can do or can not do this inside SGML applications (SGML capacity to host constraints). We will discuss ways of associating a constraint language to the SGML model. Furthermore we will try to define the scope of that language. In the end, what we are trying to do is to add a new semantic validation process to the SGML authoring and processing model. We will present also several case studies where the absence of semantic validation could be quite tragic.

Concluding this paper we will present a simple solution that implements the discussed constraint language and puts it to work with existing SGML applications (our case studies).

1 Introduction

The questions that will be addressed in this paper are very much tied up to Document essence. What is a Document? What its purpose is? Who is its target audience? Must we have some special care regarding any of these items?

We can find many sorts of answers to those questions looking to what has been written since the beginning of History. From our point of view, one definition, one aim, is common to all those texts: a Document is a written register. What is registered is a human action, a human thought, a human creation, ...

Each document has its target audience. It can be the children of the world, the women of Paris, the African people or the entire human kind. The thing to remember here is that the author had the target audience in mind when he was writing (at least in most cases).

The relevance of semantic errors in content depends on the audience. A personnel letter written to a specific person may contain semantic errors; they will affect one person. On the other hand, every error in a book written for the students of fifth grade will have a tremendous impact.

Two hundred years ago an author would write his documents, in most cases, all by himself, and would follow the process of publishing near to its end, taking as much time as he needed. Nowadays, this would be a very slow, high cost process. In the last years, Electronic Publishing became a strong industry. In order to be competitive, each stage of the process of authoring and publishing of a document was specialized and delivered to the responsibility of a specialized individual.

Somewhere in the middle of this evolution something was lost. The author only takes responsibility during the authoring stage. He won't follow his documents throughout the other stages. Many errors that could be captured during the following stages are left behind, because the person facing them does not know anything about the subject in question.

This problem gets even worse if instead of one author we have an authoring team, if instead of authoring we are transcribing, if we are doing things on a hurry.

At this stage you could say: "Relax, we have SGML!".

Yes, we do have SGML. And SGML can solve up to 50% or 60% of the problem.

With SGML we can establish rules for document production. We can enforce structural correctness. We can validate documents' structure. We can make the stages following authoring fully automatic - this way the author will be the only responsible for the document (like in the old days).

But one thing that SGML cannot give is time. And time is important. Nowadays authors are often working under strong pressures to deliver as soon as they can. Rush leads to errors. We have three kinds of errors: lexical and syntactic (related to the language - there are already tools to automatically correct them), structural (SGML tools take care of these) and semantic (related to concepts - there are no tools to deal with them).

In old times, most of the semantic errors were caught in later stages by the author that was accompanying the process. Today the author is confined to the authoring stage, losing contact with his documents after that. In big projects is worse. The author (the expert on the subject being addressed) is supervising a group of people (inexperts on the subject) that are typing the documents; this situation will lead to long and complex revision cycles.

Having this scenario in mind, it is easy to conclude that any automatic validation task will help to improve the quality of electronic publishing we have.

SGML plays an important role, it gives automatic structural validation, and in certain cases can lead an inexpert author through typing a certain kind of document.

However, in this validation universe there is a tremendous lack: semantic validation. Of all the validation tasks is the last that should be performed and the most complex and difficult to implement.

In the next sections, we will address this problem, aiming at specifying the path towards some light.

2 Case studies: What can go wrong with existing SGML applications

In this section we present two case studies that will illustrate our proposal.

These two case studies emerge from a project context where we are incharged of collecting information from various sources and making this information available through the Internet.

In both of them, there are problems that could be solved if some simple automatic semantic validation was available.

2.1 Parish Registers

In this case, our source of information is a parish archive. In catholic communities, the parish kept records of catholic acts, like marriages, births and deaths. Though we have several kinds of documents: marriage articles, baptism certificates, death certificates, and others that don't fall in any particular classification.

The aim of this project is to gather the information within those records and build a database of individuals grouped in families. Over this database, we can build statistic models to track habits, human practices, to learn a bit more about our ancestors.

We are creating a SGML document database with those documents. We have people typing our documents (the originals are quite old and confuse in order to allow an automatic upload). People often make mistakes. The SGML parser takes care of the structural/syntax ones. The others (semantic) remain and lead to several problems that will reflect in the analytical models:

- *negative ages* - probably a certificate's date was introduced with a wrong value.
- *death before baptism* - the same problem as mentioned above.

- *marriages between people with age differences higher than 100* - there is a wrong date somewhere in document related to one of the couple.

2.2 Archaeology

Another information source we have is a group of archaeologists.

They are producing SGML documents, each one reporting an archaeologic place or artefact. Each document has two special elements that are not optional: *latitude and longitude*. These elements give the geographical coordinates of the object or place in question. We want to tie each document with a point within a map building a Geographical Information System (GIS).

One thing we want to ensure is that every coordinate falls into the map, within a certain range. This of course, can not be ensured inside SGML scope but could be easily done with a simple constraint language.

3 SGML and Semantics

Can we just add constraints to SGML in order to process semantic validations? What is missing? Do we need more than just constraints?

Everyone who knows SGML, knows that we can not use its syntax to express constraints or invariants over element contents. SGML was designed to specify structure, and today its context, with all the available tools, makes it a strong and powerful specification tool.

So, if we want to be able to restrict element contents we must add something to SGML. We can just add extra syntax or we can design a completely new language that could be embedded in SGML or coexist outside.

Until now, all the constraints we feel the need to specify are quite simple. More or less we just want to restrict atomic element values, check relations between elements or performing a lookup operation of some value in some database. This happens, probably, because all the validations at higher levels are enforced by the SGML parser according to a specific DTD.

Though it seems that a simple constraint language can do the job. However, we can distinguish two completely different steps when going towards a semantic validation model:

the definition the syntactic part of the constraining model; the statements that express the constraints.

the processing the semantic part of the constraining model; the constraints interpretation.

This two steps have different aims and correspond to different levels of difficulty in their implementation.

The definition step, involves the definition of a language or the adoption of an existent one.

For the processing step, we need to create an engine with processing capabilities concerning the statements written in the above language.

Somewhere in the middle of these steps we will face the need of having typed information with all the inherent complexity.

At this moment, you could ask: "Why do we need this extra complexity? Can’t we live without types?".

Look at the following example taken from the second case study 2.2:

```
SGML Document
     ...
        <latitude>41.32</latitude>
     ...
```

```
Constraint
        latitude > 39   and   latitude < 43
```

In this example we want to ensure that every latitude value is within a certain range. We are performing a domain range validation.

We are comparing the latitude contents against numeric values, that have an inherent type (integer or float). So, the engine that will execute this comparison needs, somehow to infer the type of the element being compared.

Examples like the one presented above are quite simple. Numeric content has a more or less normalized form. But there are others, a lot more complicated, like dates - we have more than 100 formats to write a date (and probably each one of us can easily come up with some new ones).

For the time being, we envisage two solutions to deal with the problem of data normalization and type inference:

- *writing and implementing complex tools to do the job*: first the normalization and then the type inference.
- *introducing some changes into the DTD* that will take care of data normalization and will ease the task of type inference.

Obviously, the second is the one to follow, because it is simple, and because some DTD developers have already been worried with data normalization, and they have implemented some solutions.

Probably there are others, but the best I saw til now is the TEI DTD [TEI].

The solution is quite simple, they just add an attribute named *value* to elements with normalization problems. This way, the author can write those elements in the form he wants as long as he fills the *value* attribute with the normalized form.

Example:

```
... it happened in <date value="1853.10.05"> the fifth
of October of the year 1853</date> ...
```

Here, the normalized form being adopted for date elements is the ANSI format.

As we can see on the example, we can solve the normalization problem adding the *value* attribute and we can use a similar solution to take care of the type inference problem. We will add another optional attribute to all the elements, named *type*. This attribute will be filled with a string denoting the content type of the respective element.

The examples above would look like the following:

Example (latitude):

```
...
   <latitude type="float">41.32</latitude>
...
```

We can assume that when the *value* attribute is not instantiated the element content is already written in a normalized form.

Example (dates):

```
... it happened at <date type="date" value="1853.10.05">
the fifth October of the year 1853</date>
...
```

At this point we could ask the following question: Do we need to type every element? Or just the atomic ones (PCDATA)? We will answer this question in the next section.

4 Designing a Model for Constraining

The question raised at the end of the previous section is a yes or no question. But either one of the answers carries an heavy weight.

An yes answer, would mean that besides the atomic types we also must have structured types, in order to match the higher elements in the document tree. Therefore we would have a complete mapping between the document structure and the type model. The consequences of this, drawbacks and advantages, are:

- a more complicated type system.
- probably the structured types would be best inferred from the DTD than from element content, this would imply the creation of that conversion tool ([RRAH97]).
- having the whole document mapped into a type model, it becomes quite easy to create tools to process that document inside the new environment as in [RAH95].
- having a complete mapping between our document and an abstract data type model would enable us to process the document inside the abstract data type model (not the model itself but the system that supports it); this would turn possible the use and creation of other tools (non SGML), very powerful (these tools are the operators and their combinations behind the abstract data type model).

In the other hand, if we decide to type just some atomic elements (some leaves of the document tree), we could predict the following consequences:

- the constraining language would be very simple; probably restricted to atomic types and some lookup functions.
- the processing engine would turn to be very simple and easy to implement.
- the abstract model would be incomplete; we would have a set of small bits of the complete model; this would disable any manipulation of the document inside the abstract model.

In a previous work ([RRAH97]), we presented a solution within the first case. We performed a complete conversion of the document into an abstract data type model. Back then, we did not worry about data normalization and type inference. Concerning types, we just wrote a conversion schema between the DTD and the abstract data types of the system we were using to support our implementation ([ABNO97a], [BA95a]).

It will be very easy to adapt that processing model to work with the changes we proposed so far: the *value* attribute to deal with data normalization, and the *type* attribute to help the type inference.

Although it may seem very heavy, this model represents our choice because beyond constraining it will allow us to specify further processing and transformation of documents in an high level of abstraction. However, we will not give up on the second option and in the next months we will implement a new processing model that only creates abstractions for the SGML document leaves involved in the specification of constraints.

So, for the moment, the model we propose to deal with constraints is represented in figures 1 and 2.

As shown in Fig.1 we add an extra process to the SGML processing model. This new process will run an additional validation that takes care of the constraints. In practice, we have just one checking process that deals with the two validation tasks.

Figure 2 illustrates the new validation process. Both, the designer and the user must provide information to settle down this process.

Once the designer has written the DTD, he executes *dtd2cam*; this procedure takes the DTD as an argument and produces an algebraic model in CAMILA (maps each element into a type) and generates an invariant (constraint) for each type (by default all the invariants return true); those invariants are written into a file. Then the designer can edit this file and rewrite the invariants' body to meet his needs.

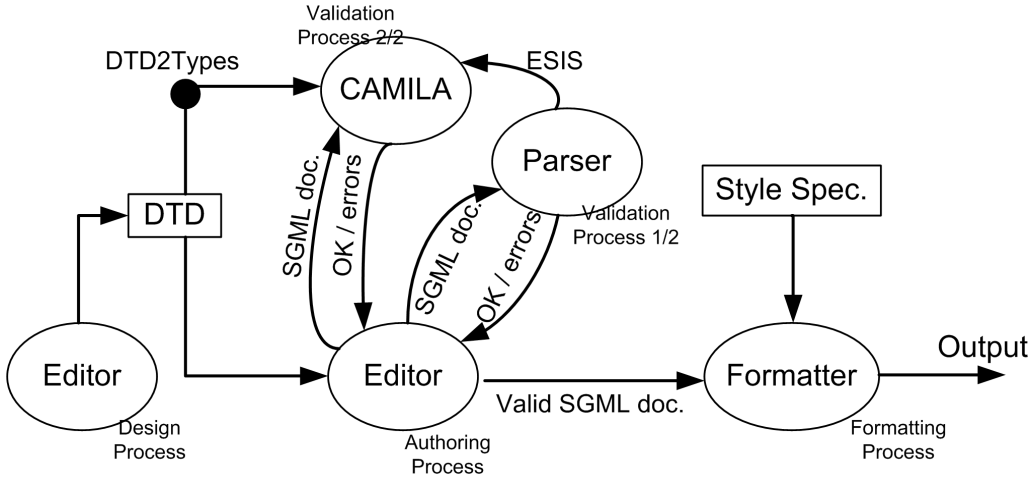

Figure 1: **Proposed SGML authoring and processing model**

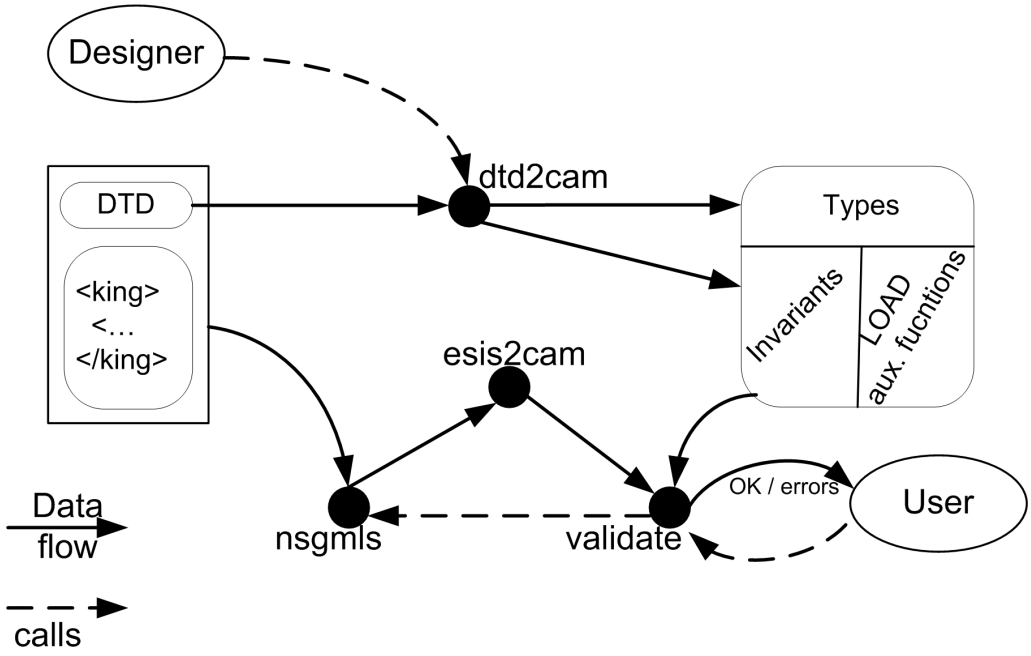

Figure 2: **CAMILA Validation Process**

From this moment the user can start authoring; when he has finished, he can run the editor's validate command which is now bound to an external *validate* function; this function calls *nsgmls* [CLA] [COV], which returns the document in ESIS format; this text is then passed to another function, *esis2cam*, which converts it into CAMILA, taking care of data normalization and dealing with some type inference; the *validate function* takes this CAMILA text together with the invariants file (described above) and checks them returning the result to the user. Notice that structural validation has been done during the execution of *nsgmls*.

4.1 Are We Reinventing the Wheel?

As everyone knows, today we can not afford the reinvention of something already discovered in the past. Therefore, looking for similar problems already solved before jumping into development of a new solution, seems a good policy.

In this particular case, we are trying to establish a path towards the implementation of semantic analysis applied to SGML documents.

Looking around inside the Computer Science area we find a good similar problem with some solutions: *Programming Languages.*

We can easily map what we are trying to do with SGML documents to what has been done with programming languages. Furthermore, we can map SGML documents to Programming Languages.

Look at the following table that is showing a comparison between these two *worlds*:

program	document instance
language	DTD
terminal symbols	SGML declaration
grammar rules	DTD

We can look at a SGML DTD as if it is a formal grammar. From there is easy to conclude the matching that is showed in the above table.

As the DTD is the heart of an SGML document grammars are the heart of programming languages.

In the beginning, programming languages had to be processed in order to obtain the runnable code corresponding to the program. The processing of a program, compilation, comprised the following steps:

- Lexical Analysis
- Syntactic Analysis
- Semantic Analysis
- Code Generation

Back then, the model used to process programs was the so called Syntax Directed Translation (SDT). In this model, the lexical analysis became automatic, the syntactic analysis also became automatic but, the semantic analysis did not. Semantic Analysis was programmed by the author of the language in the host language (the language that implemented the tools being used).

Looking again into the SGML world that is what we have. The SGML parser performs the lexical and syntactic analysis and the semantic analysis is left to be programmed in an external tool or programming environment.

In 1968, Donald Knuth introduced a new approach: *Attribute Grammars* ([KNUTH68]).

With *Attribute Grammars* we still had the simplicity of *Context-free Grammars*, but we now had the attributes to specify semantics. This extended formalism gave birth to a new model of compiler construction (programming language processing) designated *Semantics Directed Compiler Generation* [Tie80].

In the early 80's they had the same situation we have now in this work. They had a formalism to specify semantics and they needed an engine to process it. In the next years some engines were developed with some success, like the one we are using in most of our projects: the *Synthesizer Generator* [RT89a]

We felt the need to add semantics to SGML documents. We are doing it through the definition of constraints. In this paper, we presented a solution to deal with data normalization and type inference. We will add attributes to SGML elements to specify part of the semantics. We did not formalize the specification of a language for semantics but from the examples we can conclude that that is not the problem. The problem will be the processing of those constraints.

We feel that this approach that we are trying to follow is very close to the Attribute Grammars approach. The problems we are facing have been already faced by them in the past.

Perhaps the solutions found for the Attribute Grammars can help us solve our problems. But, for now, regarding this parallelism we feel we are in the right track.

4.2 Future Work

In a previous work [RRAH97], presented at the SGML/XML'97 conference, we used an external system with its own language to define and process constraints. In the near future we will define a specific language for specifying constraints to be associated with SGML documents, probably in a syntax close to the one that SGML users are used to.

We are integrating this semantic validation scheme into an environment we are developing for Document Programming. This environment ([LRH97]) called INES will produce specific structured syntax directed editors, with formatting capabilities.

We hope that we will help to improve document quality in the future.

5 Aknowledgements

Thanks are dued to JNICT and PRODEP for the grant under which this work is being developed.

References

[McGrath98] Sean McGrath, *"Parseme.1st"*, Prentice Hall, 1998.

[RH98] Ramalho, J. C., and Henriques, P. R., *"Beyond DTDs: constraining data content"*, In proceedings of "SGML/XML Europe 98" conference, Paris, May 1998.

[ABNO97a] J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira, *CAMILA: Formal Sofware Engineering Supported by functional Programming.* In A. De Giusti, J. Diaz, and P. Pesado, editors, Proc. II Conf. Latino Americana de Programacion Funcional (CLaPF97), pages 1343-1358, La Plata, Argentina, October 1997.

[BA95a] L. S. Barbosa and J. J. Almeida, *CAMILA: A Reference Manual* Technical Report DI-CAM-95:11:2, DI (U.Minho), 1995.

[Bra96] N. Bradley, *The Concise SGML Companion*, Addison-Wesley, 1996.

[CLA] James Clark, *Sp: An sgml system conforming to international standard iso 8879*, `http://www.jclark.com/sp/index.htm`.

[COV] Robin Cover, *Sgml parsers* `http://www.sil.org/sgml/publicSW.html#parsers`.

[Her94] Eric van Herwijnen, *"Practical SGML"*, Kluwer Academic Publishers, 1994.

[KNUTH68] Donald E. Knuth, *"Semantics of context-free languages"*, Mathematical Systems Theory, 2(2):127–145, 1968.

[LRH97] A. R. Lopes, J. C. Ramalho, and P. R. Henriques, *Ines: Ambiente para Construção Assistida de Editores Estruturados Baseados em SGML*. In Simpósio Brasileiro de Linguagens de Programação, Universidade de Campinas, Campinas, S. Paulo, Brasil, Sep. 1997.

[Oli90] J. N. Oliveira, *A Reification Calculus for Model-Oriented Software Specification*, Formal Aspects of Computing, 2:1-23, April 1990.

[Oli92] J. N. Oliveira, *Software Reification Using the Set Calculus* , In Proc. of the BCS FACS 5th Refinement Workshop, Theory and Practice of Formal Software Development, London, UK, pages 140-171, Springer-Verlag, 8-10 January 1992 (Invited paper).

[RAH95] J. C. Ramalho, J. J. Almeida, and P. R. Henriques, *David: Algebraic Specification of Documents*, In A. Nijholt, G. Scollo, and R. Steetskamp, editors, TWLT10 - Algebraic Methods in Language Processing - AMiLP95, number 10 in Twente Workshop on Language Technology, Twente University - Holland, Dec. 1995.

[RRAH97] J. C. Ramalho, J. G. Rocha, J. J. Almeida and P. R. Henriques, *SGML Documents: where does quality go?*, In SGML'97 Conference Proceedings, Washington - USA, Dec. 1997.

[RT89a] Thomas Reps and Tim Teitelbaum, *"The Synthesizer Generator: A System for Constructing Language-Based Editors"*, Texts and Monographs in Computer Science, Springer-Verlag, 1989.

[RT89b] Thomas Reps and Tim Teitelbaum, *"The Synthesizer Generator Reference Manual"*, Texts and Monographs in Computer Science, Springer-Verlag, 1989.

[TEI] C.M. Sperberg-McQueen and Lou Burnard, *Guidelines for Electronic Text Encoding Interchange (TEI P3)*, Chicago: ACH/ACL/ALLC, 1994.

[Tie80] Martti Tienari, In book *Semantics Directed Compiler Construction*, Lecture Notes in Computer Science 94, Springer-Verlag, 1980.