# SGML Documents: Where Does Quality Go?

José Carlos Ramalho
jcr@di.uminho.pt

Jorge Gustavo Rocha
jgr@di.uminho.pt

José João Almeida
jj@di.uminho.pt

Pedro Henriques
prh@di.uminho.pt

Departamento de Informática
Universidade do Minho
Braga – Portugal
October 23, 1997

**Abstract**

Quality Control in Electronic Publications should be one of the major concerns of every project.

Big projects try to gather information from a series of different sources: universities, libraries, museums and other scientific or cultural organizations.

Collecting and treating information from several different sources raises a very interesting problem: the assurance of quality.

Quality in Electronic Publications can be reflected in several forms, from the visual aspects of the interface and linguistic/literary ones to the correctness of data. We are concerned with the lowest boundary of this spectrum, correctness of data.

With SGML we can solve a small part of the problem, structural correctness. SGML provides a nice way to structure documents keeping a complete separation between structure (syntax) and typesetting. Today there are lots of editors and environments that can assist the user producing well-formed SGML documents (validating their structure). But, there is clearly a lack for content validation. There are situations where pre-conditions over the information being introduced should be enforced in order to prevent the user from introducing erroneous data; we shall call this process data semantics validation. In SGML is not possible to implement this process.

We will discuss an adaptation of the SGML syntax that will enable us to express constraints to allow some semantic validation when authoring.

Furthermore, we will propose a new SGML processing model capable of dealing with this extension. This model will be built extending the existing one. So, we will not restrict any SGML capability, instead we will add new ones.

Both, the SGML extension and the model extension, will be defined and implemented resorting to algebraic specification (SET theory and

functional programming).

# 1 Introduction

In previous work [RAH95] we have suggested an algebraic approach to document processing. This first proposal was further explored in [RAH96].

The use of data types to express documents' syntax, would enable us to reason about structured texts as we are used to about any other objects; thus we can archive documents in an internal format according to the mathematical model chosen to implement their types, and moreover we can transform documents by means of operations over document types. Those operations can then be implemented by functions as usual in model-based algebraic specification and development method. Documents are seen as algebraic terms.

In this article our main concern will not be the definition and implementation of functions for document processing (production, or transformation), but we will emphasize the clean and easy way how we can express and deal with document semantic validation in order to assure the quality of documents (at least, from the content correctness point of view).

We intend to merge the traditional SGML processing model with the algebraic approach to keep all the functionality available in the context of SGML based environments but enhance such environments with the capability of making some semantic checking during authoring. This enhancement aims at the elimination of errors (not only structural, but mainly semantics) during information manipulation in order to improve the quality of document processing tasks.

As in the past, we use Camila (a specification language and prototyping environment[ABNO97] developed at Universidade do Minho, by the Computer Science group) to implement the above mentioned validation scheme.

Section 2 is dedicated to raise the problem and to give motivation. In sections 3 and 4 we propose a solution along with its implementation. Finally, we conclude the paper with section 5 binding this article to other work in progress.

To understand the ideas discussed in this paper, the reader is supposed to be familiar with SGML, the standard mark-up language for document processing, and its intrinsic concept of *document type definition* (DTD). Concerning these matters we address the reader to [Her94, Bra96].

It is also convenient to have some knowledge about *model based algebraic approach* to software specification and development [Oli90, Oli92]. Camila is a language and prototyping environment well founded in the set-theory, not very different from Z [Spi89], Raise [Geo91], or other VDM [Jon86] inspired methods. The language is introduced in [BA95b, BA95a]. The paper [RAH95] also contains an appendix dedicated to that method.

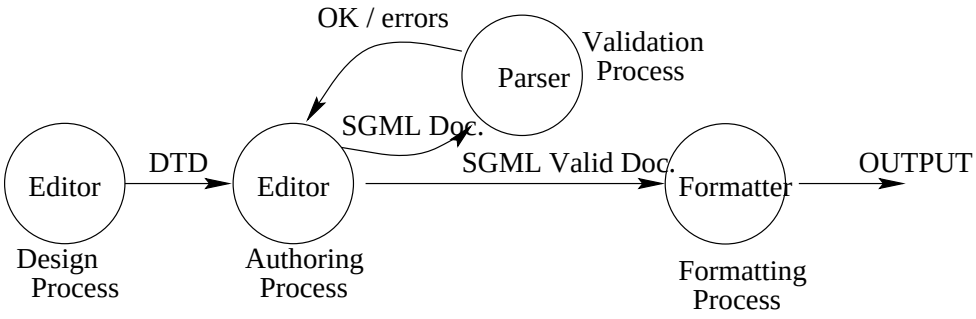


Figure 1: SGML authoring and processing model

## 2 SGML and Semantic Validation

The introduction of SGML brought new ways of working in document processing: automated structural validation, validation process during editing (the author is always aware of his document's structural correctness), platform and application independence.

Big corporations, producing large amounts of documentation, can have authoring teams working coherently.

This model of SGML documentation processing is shown in Fig. 1.

The model is sufficient for most cases but when dealing with certain information there is a lack: semantic validation! For example, suppose someone is editing a CD-ROM with History content ; an error on a king's birth date or a wrong association of kingdoms with the dates, can make all the project collapse.

Concerning consistency and correctness verification issues, SGML enforces a complete structural checking and some semantic validation (through the use of attributes); it is possible to secure that some text fields have only a restricted set of values. But, when producing certain kinds of documents we need to impose stronger constraints on data; this requirement is beyond SGML scope. We will not propose a solution for the complete problem of semantic validation only for a subset: each PCDATA is, normally, too general for the intended element type; compound content models, can have relations between some of their subelements that should be guaranteed; this kind of relations/restrictions can be expressed associating a type with each DTD's element and imposing an invariant property over that type.

In the next example we illustrate the idea.

**Example 1: [The need for semantic validation]**
An editor is editing a book on portuguese literature (using SGML). One of the chapters will focus on authors. Its main body could be defined as:

```
...
<!ELEMENT authlist - - (author+)>
<!ELEMENT author   - - (name,birthdate,deathdate,...)>
```

...

This DTD fragment states that a part of the referred chapter should describe a list of authors, each one characterized by a name, a birth date, a death date, and some other elements.

Structural validation will be ensured by the SGML parser. However, beyond syntactic correctness it is also important to state an *invariant* that should be preserved: the deathdate field for each author should always contain a value higher then the birthdate field.

---

The small example above gives a feeling of the kind of problems we found in practice, when dealing with archaeological and historical documents. The resolution of those problems seemed to be crucial to secure error free information.

In the next sections we will discuss an extension to the SGML processing model capable of dealing with these and similar validation problems.

# 3 SGML and Constraints

In order to guarantee the preservation of some semantic characteristics of documents we need to associate constraints to the DTD's element. The power to write the constraints, or type invariants, is given to the designer that will write them along with the DTD. There are two ways to enclose invariants in the DTD:

- special *comment sections* where the invariants could be written, mixed with the DTD declarations).

  **Example 2: [DTD declarations mixed with invariants]**

```
<!DOCTYPE king [
<!ELEMENT king - - (name, coname, bdate, ddate, decree*)>
<!-- INV
  inv_king(k)= ...
-->
```

---

- an anchor to an external file where the invariants will be written; the anchor will be placed in a special *comment section*.

  **Example 3: [invariants binded with an anchor]**

```
<!-- INV: king.cam -->
<!DOCTYPE king [ ...
```

The use of *comment sections* was reinforced so that new additions do not affect SGML syntax. This way, we can still use SGML parsers as they are to run the structural validation process.

From the two proposed approaches we chose the second. The first one could lead to heavy and hard to read DTDs. The second maintains the DTD conciseness and gave us one other advantage. We wrote a small compiler that giving the DTD generates a skeleton for the invariants file; this process helps the document designer making his work faster and more secure.

In order to be able to speak about SGML elements and state constraints about them, the natural choice will be a model based specification language. As shown in [RAH95], each element definition of the DTD has an implicit model (a type), and each element instance can be mapped into an algebraic expression over that model. We use an automatic conversion tool to translate the DTD into the implicit model in SET specification language Camila [BA95b, BA95a]. After this mechanical conversion, the designer will be able to associate each DTD element type with an invariant (predicate) that should always evaluate to true; if in some document the invariant evaluates to false, its author will receive an error message.

Each invariant is defined by a set of pairs formed by a condition and respective reaction.

Consider the example below that clarifies our proposal.

**Example 4: [Kings and decrees]**

The following DTD was written to define a format for editing lists of decrees proclaimed by some king.

```
<!DOCTYPE king [
<!ELEMENT king - - (name, coname, bdate, ddate, decree*)>
<!ELEMENT decree - - (date,body)>
<!ELEMENT (name, coname) - - (#PCDATA)>
<!ELEMENT (bdate, ddate, date) - - (#PCDATA)>
<!ELEMENT (org) - - (#PCDATA)>
<!ELEMENT body - - (#PCDATA | org)* >
]>
```

The text below lists some decrees proclaimed by D. Dinis, according to the DTD above.

```
<king>
    <name>D.Dinis</name>
    <coname>Farmer</coname>
    <bdate>1270.09.23</bdate>
    <ddate>1370.09.23</ddate>
    <decree>
      <date>1300.07.15</date>
      <body> From this day only bicycles are allowed to circulate in the
town of Braga.</body>
```

```
    </decree>
    <decree>
      <date>1297.07.15</date>
      <body>McDonald's will sell green wine instead of COCA-COLA.</body>
   </decree>
</king>
```

Observing the DTD and its instance it is easy to identify some invariants that should be true in every document written according to this DTD:

- the document's `date` should be higher than king's birth date (`bdate`).
- the king's `name` should exist in some database.

To formalize these invariants we want to preserve in each document, we add to the DTD a special comment section with an anchor to a file where the invariants will be maintained, as shown below.

```
<!-- INV: king.cam -->
<!DOCTYPE king [ ...
```

Each invariant will be a set of pairs *condition-reaction* (action taken by the checker if the condition – negation of the invariant – evaluates to true).

```
inv_king(k)=
   {if(k notin famous_personsDB -> k ++ " must be inserted in FPDB") ,
    if(bdate_(k) > ddate_(k)    -> k ++ " died before he has born") ,
    if(ddate_(k)-bdate_(k)> 120   -> k ++ " lived more then 120 years") ,
    if(!all(x <- decree_1(k): bdate_(k) < date_(x) /\ date_(x)  < ddate_(k) )
                                -> k ++ " made a decree outside his life")
   } ;
```

Suppose we had the following values:

- k = "D. Dinis"
- bdate(k) = "1265/06/24"
- ddate(k) = "1211/04/12"

This set of values would trigger the invariant

```
if(bdate(k) > ddate(k))
```

and would produce as result the concatenation ("++") of the king's name ("k") with the string "died before he has born".

---

After the explanation of our adaptation of SGML DTDs to accommodate invariants intended as contextual conditions that preserve documents' semantic, we will discuss in the next section how we extend the processing model to cope with this semantic verification.

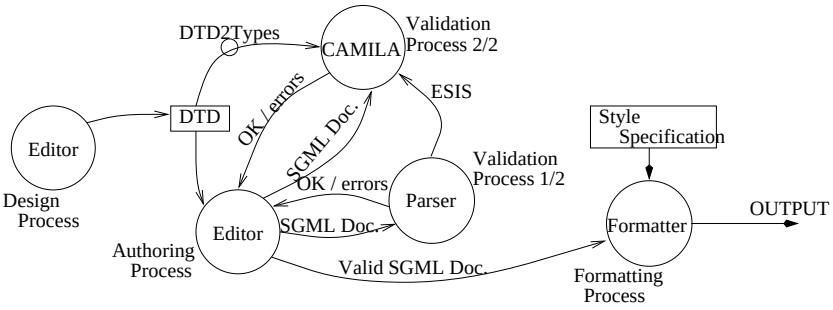


Figure 2: New SGML authoring and processing model.

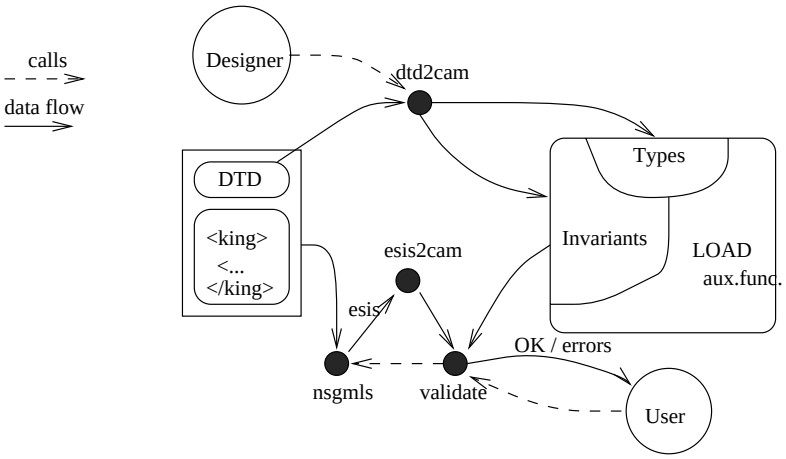


Figure 3: Camila Validation Process.

# 4 Implementation

As shown in Fig.2 we add an extra process to the SGML processing model. This new process will run an additional validation that takes care of the invariants. In practice, we have just one checking process that deals with the two validation tasks.

Figure 3 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 for each type (by default all the invariants return true); those invariants are written into a file which name came from the invariant anchor in the DTD (see previous section); then the designer can edit this file and rewrite the invariants' body to meet his needs.

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; 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.

The process whose behavior was described above comprises two generation tasks.

The first is solved with a simple compiler, esis2cam, that converts ESIS output from nsgmls into Camila, i.e., takes a document after passing the structural validation and passes it to Camila that will run the semantic validation. The ESIS format is the most used intermediate representation for SGML documents. Each line of data in ESIS represents a single "instruction" returned from the parser. The instructions most commonly encountered are:

- start ("(gid") and end (")gid") the element with generic identifier "gid"
- attribute value ("Avalue")
- data content ("-data")

Applying nsgmls to example 3 we would get the following ESIS output:

```
(king
(name
-D. Dinis
)name
...
(decree
(date
-1300.07.15
)date
(body
-From this day only ...
)body
)decree
...
)king
```

The second task (dtd2cam) is not so simple because it involves a mapping between a DTD and an algebra:

- the DTD is translated into a model-set
- each element is mapped into a type according to a predefined translation scheme [RAH95, RAH96]
- each element will have an invariant
- each invariant is a (constraint - reaction)-set, by default it will be the true value. The constraint and the reaction are written according to Camila syntax with Camila operators.

The following example gives a clear idea of the process. The DTD that will be used is the one presented earlier of "kings and decrees".

**Example 5: [Execution of dtd2cam]**

The DTD has the following declaration:

```
<!ELEMENT king - - (name, coname, bdate, ddate, decree*)>
```

dtd2cam generates the following Camila code:

```
TYPE
    king=name_:name
    coname_:coname
    bdate_:bdate
    ddate_:ddate
    decree_l:decree-seq
    ;
ENDTYPE
inv_king(k)=true;
...
```

---

# 5 Conclusion

In this paper we have presented and discussed a proposal for an extra validation process to be added to the SGML processing model. This new validation process enables us to put some kind of data constraints in the DTD. Then it becomes possible to restrict the values of some elements of documents to certain ranges and also to enforce some relationship between text elements. This way many errors given by a distracted author can be avoided.

We think that this validation scheme can help to improve information quality. This parameter is crucial when distributing large amounts of information to large amounts of users, for example in the Internet.

In every real example we have tried so far, we didn't found the need to impose highly complex invariants. This is probably due to SGML since it already enforces a structural validation. Most of the invariants we need to write, are validations of data atomic types like strings and numbers and relations between them. This leads to the feeling that our model can be simplified and optimized. One way to do this is to replace Camila with a simpler constraint language (which is being done already).

We are joining this validation scheme with our document management environment INES [LRH97]. The combination of the two will result in a more powerful processing environment with an high level of quality in the information being produced.

## Acknowledgments

We would like to thank the precious comments and suggestions from our anonymous referees that were very helpful to improve this paper.

We also are grateful to Luis Barbosa for the profitable discussions.

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

## References

[ABNO97] J. J. Almeida, L. S. Barbosa, F. L. Neves, and J. N. Oliveira. CAMILA: Formal software engineering supported by functional programming. In A. De Giusti, J. Diaz, and P. Pesado, editors, *Proc. II Conf. Latino Americana de Programacin Funcional (CLaPF97)*, pages 1343–1358, La Plata, Argentina, October 1997. Centenario UNLP.

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

[BA95b] L. S. Barbosa and J. J. Almeida. Systems Prototyping in CAMILA. Bristol Lecture Notes DI-CAM-95:11:1, 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.

[Geo91] C. George. The RAISE specification language: a tutorial. In *Proc. of VDM'91*. LNCS (551), 1991.

[Her94] E. Herwijnen. *Practical SGML*. Kluwer Academic Publishers, 1994.

[Jon86] C. B. Jones. *Systematic Software Development Using VDM*. Series in Computer Science. Prentice-Hall International, 1986.

[LRH97] A.R. Lopes, J.C. Ramalho, and P.R. Henriques. Ines: Ambiente para construo assistida de editores estruturados baseados em sgml. In *Simpsio Brasileiro de Linguagens de Programao*, Universidade de Campinas, Campinas, S. Paulo, Brasil, Set. 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.

[RAH96] J.C. Ramalho, J.J. Almeida, and P.R. Henriques. Document semantics: two approaches. In *SGML'96: Celebrating a decade of SGML*, Sheraton-Boston Hotel, Boston, USA, Nov. 1996.

[Spi89] J. M. Spivey. *The Z Notation — A Reference Manual.* Series in Computer Science. Prentice-Hall International, 1989.