SGML Documents: Where does Quality Go?

José Carlos Ramalho Jorge Gustavo Rocha José João Almeida
Pedro Rangel Henriques

1998

Abstract

Quality control in electronic publications should be one of the major concerns of everyone who is managing a project.

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

Collecting and treating information from several different sources raises very interesting problems, one being the **assurance of quality**.

Quality in electronic publications can be reflected in several forms, from the visual aspects of the interface, to linguistic and literary aspects, to the correctness of data.

With SGML we can solve part of the problem, structural/syntactic correctness. SGML provides a nice way to specify the structure of 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 and valid SGML documents (validating their structure). However, current software still gives the user too much freedom. The user has full control of the data being introduced, creating a margin for errors. In this context 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.*

The idea is to constrain the values of some structural elements of a document according to its final purpose. This way the user (who writes the documents according to that DTD) will not have full control of his data; he will be forced to obey certain domain range limitations or certain information relationships.

SGML does not have the necessary constructs to implement this extra validation task.

In this paper we will present and discuss ways of associating a constraint language with the SGML model. We will present the steps towards the implementation of that language. In the end, we present a new SGML authoring and processing model which has an extra validation task: semantic validation.

Along the paper we will show some case studies that could have their quality improved with this new working scheme.

1 Introduction

The questions that will be addressed in this paper are very much tied up to the essence of documents. What is a document? What is its purpose? Who is its target audience? Must we take 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. In our point of view we can say that one definition, one aim is common to all those writings: a document is a written register of some human action, some human thought, some 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 (at least in most cases) the author had that audience in mind when he or she was writing.

The relevance of errors in document content depends on the audience. A personal 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

fifth-grade students of a nation will have a tremendous impact. Any error in a nuclear power plant manual could also have very tragic consequences.

There are three kinds of errors: lexical (related to the language vocabulary - there are tools to automatically correct these); syntactic/structural (SGML tools take care of these); and semantic (related to concepts and reality - there are no appropriate tools to prevent or correct these). With SGML we can solve a major part of the error pruning problem. We can establish rules for document production. We can enforce structural correctness (validating document structure). We can make the stages following authoring fully automatic, so as to make the author the only person responsible for his or her documents.

At this point, it is easy to conclude that any automatic semantic validation tool will help to improve the quality of electronic publishing we have. Of all the validation tasks this is the last to be performed and the most complex and difficult to implement. Throughout this paper we will describe the steps towards such an implementation.

This work is in some ways the conclusion of some previous proposals reported in [RAH95, RAH96, RH98].

2 SGML and Semantic Validation: What can go wrong with existing SGML applications

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.

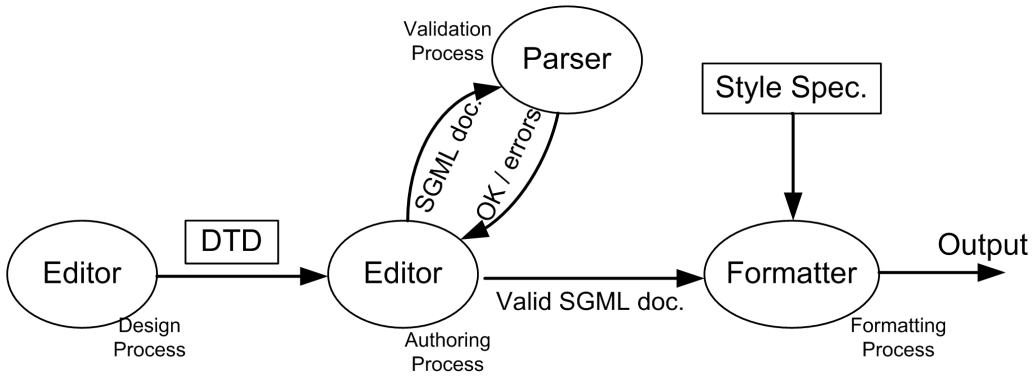

Figure 1: **SGML Authoring and Processing Model**

The model is sufficient for most cases but when dealing with certain information there is a serious gap: semantic validation! For example, suppose someone is editing a CDROM with historical material; an error on a king's birth date or a wrong association of kingdoms with the chronology can make the entire 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 the scope of SGML. We will not propose a solution for the complete problem of semantic validation; our solution deals only with a subset of validation problems: constraining the `PCDATA` content within a given element type, and checking the relationships between sub-elements or attributes.

In the next subsections we present two case studies to illustrate these ideas.

These two case studies emerge from a project context where we are trying to collect information from various

sources and make 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 is a parish archive. In Catholic communities, the parish kept records of Catholic sacraments, like those accompanying marriages and deaths. Thus 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. Using this database, we can build statistic models to track habits, human practices, to learn a bit more about our ancestors.

We are creating an SGML document database with those documents. We have people typing our documents (the originals are quite old and too irregular to allow an automatic upload). Transcribers often make mistakes. The SGML parser takes care of the structural/syntax ones. The others (semantic) remain and lead to several problems that will have some influence in the analytical models; for example:

- 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 some document related to one of the couple.

All these problems could be solved with some simple domain constraints like: birth date should always be lower than death date; concerning marriages, the difference between birth dates of the two people should always be less than a certain value, etc.

2.2 Archaeology

One other source of information that we have is a group of archaeologists with whom we are working.

They are producing SGML documents, each one reporting an archaeological site 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 Constraints

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 cannot use its syntax to express constraints or invariants over element content. SGML was designed to specify structure, and for structural constraints the large set of available tools makes it a strong and powerful specification tool.

So, if we want to be able to restrict element content 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. Adding extra syntax would imply the modification of every existing tool so they could recognize the extension. The cost of this

solution is extremely high so we will choose the other: the definition of a language that enables the specification of constraints over SGML elements.

Thus far, all the constraints we have felt the need to specify have been quite simple. For the most part we just want to restrict atomic element values, check relationships between elements, or perform a lookup operation of some value in a database. The requirements are probably so simple primarily because all the more complex constraints are expressed in the DTD and are enforced by the SGML parser.

Thus 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.

These 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 the ability to evaluate the statements written in the constraint language.

To implement this engine we will face the need of having typed information with all its inherent complexity.

At this moment, the reader may ask: "Why do we need this extra complexity? Can't we live without types?"

Consider 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, more complex, 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). In another case study we are dealing with 12th-century documents; in those documents `"Afonso Henriques"` the first portuguese king, has his name written in two different ways: `"Afonso"` and `"Affonso"`. Other characters are referenced in more than two different ways.

We shall call this issue the normalization problem. Normalization is critical when you want to build indexes or information retrieval engines over the documents. The computer must understand that some different references correspond to the same object.

For the time being, we envisage two solutions to deal with the problems 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 type inference task.

Obviously, the second is the one to follow, because it is simpler, and because some DTD developers have already been worried with data normalization, and they have adopted some solutions in this approach. Probably there are others, but the best we found till now is the TEI DTD [SB94]. The solution is quite simple; they just add an attribute named *value* to elements with normalization problems. This way, those elements can take any desired form, as long as the *value* attribute carries the normalized form.

Example:

```
... it happened on <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 one can see by 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.

Following this idea the examples above would look like:

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

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

names ... king <name value="Afonso">Affonso</name> proclaimed several decrees ...

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

At this point we could ask: *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 answer carries a heavy weight.

A 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:

- The type system grows more complicated.
- The structured types would be best inferred from the DTD instead of the element content; this would imply the creation of some conversion tool.
- A complete mapping between a 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 make possible the creation and use of other very powerful non-SGML tools (these tools are the operators and their combinations behind the abstract data type model), as in [RAH98].

On 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 constraint language would be very simple; 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 ([RAH98]), we presented a solution within the first of these directions. 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 besides constraints 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; in the next sections we will present a prototype environment that implements a processing model that only creates abstractions for the SGML document leaves involved in the constraints specification.

So, for the moment, the model we propose to deal with constraints is represented in the following figure.

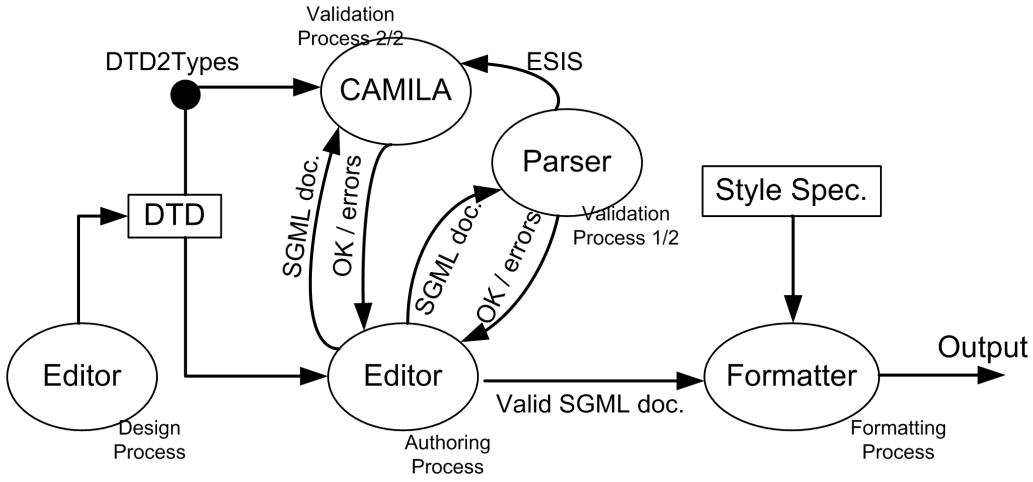

Figure 2: **New SGML authoring and processing model**

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

In the next section we will take a detailed look at the implementation of this new process and at its consequences in the life cycle of document production.

5 Implementing a New Model

We are now going to specify the steps towards an implementation of the model presented (figure 2).

The new prototype will demand new capabilities from both people and software. We can think of these in a two-layer model: the top one relates to people, how they will coordinate their work, DTD designers and authors/users; we shall call it the operating layer. The lower one relates to software routines, how they will coordinate the work and interface between them; we shall call it the software layer.

Let's take a deeper look into them:

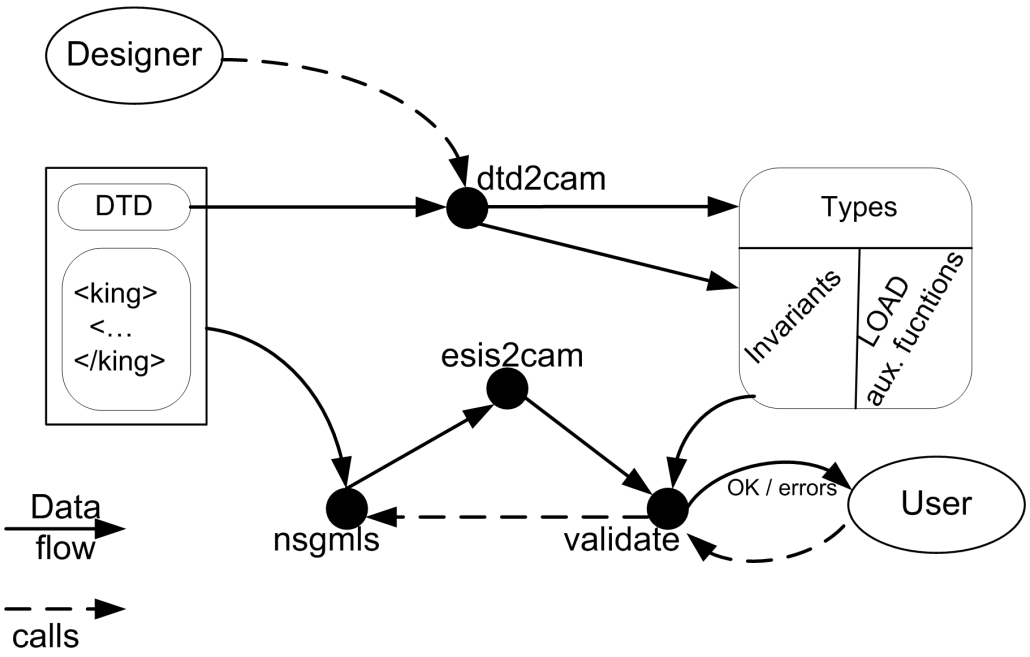

Figure 3: **CAMILA Validation Process**

Operating layer We consider the intervention of two persons: the designer and the author. In the old model 1, the designer only has to write the DTD, whereas in this model we must also write the constraints along with the DTD. At the end of the design process a copy of the DTD is sent to the authoring process and the same copy *together with the constraints* is sent to a new piece of software that will perform the additional validation task (see figure 2). The author/user will work in the usual way (the changes are invisible from the user point of view).

Software layer This layer is composed of several pieces, one major one and other smaller ones (see figure 3). We named the major one "CAMILA validation process on behalf of the environment that is hosting/implementing this process [ABNO97a]. The others are named according to their functionality: DTD2CAM, ESIS2CAM and Parser (nsgmls - [Clark]). All the process is centered in the major CAMILA process, which receives three inputs and produces one output. The DTD produced in the design process is sent to DTD2CAM, which translates the elements with associated constraints into CAMILA objects and sends these to the CAMILA process. The constraints resulting from the design process are sent directly to the CAMILA process. On the other end, when the user wants to validate the document after editing, the user activates the validation task that will send the document to a normal parser; the output of the parser, in ESIS format, is sent to ESIS2CAM, which translates the elements with associated constraints into CAMILA objects. The CAMILA process, given these three inputs, can evaluate the constraints and give feed-back to the user.

It is easy to notice that this is not a simple process and that several decisions were made in the course of implementation. In the next subsection we will trace the path of an example passing through the system.

6 Associating constraints with elements

In order to guarantee the preservation of some semantic characteristics of documents we need to associate constraints with the DTD's elements.

The power to write the constraints is given to the designer, who will write them along with the DTD. Here, in terms of implementation we had a choice to make: we could enclose constraints in the DTD in three different ways:

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

  Example: **DTD declarations mixed with constraints**

```
<!DOCTYPE king [

<!ELEMENT king  - - (name, coname, bdate, ddate, decree*)>
<!-- Constraints
    king(k) = ...
-->
...
]>
```

- special *processing instructions* where the constraints could be written, mixed with DTD declarations.

  Example: **DTD declarations mixed with constraints** <

```
<!DOCTYPE king [

<!ELEMENT king  - - (name, coname, bdate, ddate, decree*)>
<?Constraints
    king(k) = ...
>
...
]>
```

- an anchor to an external file where the constraints will be written; the anchor is placed inside a special *comment section*.

  Example: **Constraints bound with an anchor**<

```
<!-- Constraints: 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 and to keep the compatibility with existing applications.

From the three proposed approaches we chose the third. The first two, constraints mixed with DTD declarations, would lead us to heavy and hard to read DTDs since some DTDs are already too complex. The third approach maintains the DTD's concision and enabled us to build another small but handy tool: a small compiler that accepts the DTD as input and generates a skeleton for all the constraints; this process helps the document designer to make his work faster and more secure.

7 An example

Let's now describe an example to guide us through the rest of the discussion. The actual case studies we have in hand are quite complex and each one has its own very specific problems. So we created a smaller problem with enough complexity to illustrate our ideas and our prototype.

Each one of our documents will be a list of decrees proclaimed by some king. It is a simple version of one of our case studies.

Kings and decrees (DTD)

```
<!DOCTYPE king [

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

<!ELEMENT (name, coname, bdate, ddate, date) - - (#PCDATA)>
<!ATTLIST bdate value CDATA #IMPLIED
                type  CDATA #FIXED date>

<!ATTLIST ddate value CDATA #IMPLIED
                type  CDATA #FIXED date>

<!ATTLIST date  value CDATA #IMPLIED
                type  CDATA #FIXED date>

<!ELEMENT body - - (#PCDATA)>

]>
```

Notice the attributes `value` and `type` defined for date elements. The `type` attribute has a fixed value that will be used later to infer the type of the element. The `value` attribute will be used to normalize dates; the processor will use the value of this attribute instead of the element content.

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

D. Dinis' decrees (dinis.sgm)

```
<king>
   <name>D. Dinis</name>
   <coname>Farmer</coname>
   <bdate value="1270.09.23">23 Sep 1270</bdate>
   <ddate value="1370.09.23">23 Sep 1270</ddate>

   <decree>
      <date value="1300.07.15">the fifteenth August of
the year 1300</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 we can think of some properties that should be verified in every document written according to this DTD:

- The decree's date should be always higher than king's birth date (`bdate`).
- The king's name should exist in our Famous Persons Database.

To specify these properties we want to preserve in each document, we add to the DTD a special comment section with an anchor to a file where those properties/constraints will be written.

```
<!-- Constraints: king.cam  -->

<DOCTYPE king [ ... ]>
```

8 Expressing the constraints

At this point, one question emerges. How will we write the constraints? In what language?

As discussed in the beginning we have two options: designing a new language or using an existing one. Since what we are developing is a prototype and the use of an existing language would save us some work, we took this option.

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 ([ABNO97a, BA95a]). After this mechanical conversion, the designer will be able to associate each DTD element type with a constraint (predicate) that should always evaluate to true; if in some document the constraint evaluates to false, its author will receive an error message.

Each constraint is defined by a set of pairs formed by a condition and respective reaction. In the example we are tracing we could write the following constraints associated with `king` element.

Specification of Properties in CAMILA (king.cam)

```
king(k) =
   { if( k notin famous-personsDB
        -> name_(k) ++ "must be inserted in FPDB"),

     if( bdate_(k) > ddate_(k)
        -> name_(k) ++ "died before he has born"),

     if( ddate_(k) - bdate_(k) > 120
        -> name_(k) ++ "lived more then 120 years"),

     if( !all( x <- decree_l(k): bdate_(k) < date_(x)
          /\ date_(x) < ddate_(k) )
        -> name_(k) ++ "made a decree outside his life" )
   };
```

Suppose we had the following values:

- $k = (\text{"}D.Dinis\text{"}, \text{"}Farmer\text{"}, \text{"}1265.06.24\text{"}, \text{"}1211.04.12\text{"}, ...)$
- $bdate_(k) = \text{"}1265.06.24\text{"}$
- $ddate_(k) = \text{"}1211.04.12\text{"}$

This set of values would trigger the constraint `if( bdate_(k) > ddate_(k) )` and would produce as result the concatenation (`++`) of the king's name ($name_(k)$) with the string `"died before he was born"`.

9 Extending the processing model

After the explanation of our adaptation of SGML DTDs to accommodate constraints intended as contextual conditions that preserve document semantics, we will discuss now how we extend the processing model to cope with this semantic verification.

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 constraints.

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 a default constraint for each type (by default all the constraints return true); those constraints are written into a file which name came from the constraint anchor in the DTD; then the designer can edit this file and rewrite the constraints 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` [Clark], which returns the document in `ESIS` format; this text is then passed to another function, `esis2cam`, which converts it into CAMILA; the `validate` function takes this CAMILA text together with the constraints 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 so that it can 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 ("$Anametypevalue$")
- data content ("$-data$")

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

```
(king
(name
-D. Dinis
)name
...
(decree
Avalue CDATA 1300.07.15
Atype CDATA date
(date
-the fifteenth August of
the year 1300
)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, RAH98]
- each element will have an associated constraint
- each constraint is a condition—reaction tuple, by default it will be the *true* value. The condition and the reaction are written according to CAMILA syntax with CAMILA operators.

Let's apply this to our example to get a clear idea of the process.

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_:date
        ddate_:date
        decree_l:decree-seq

        ;

ENDTYPE

    king(k)=true;

    ...
```

We have traced the proposed example through our entire model, showing partial results of all the tasks involved. We hope this helped to clarify our ideas and our new SGML authoring and processing model.

10 Conclusion and Future Work

Our main concern in the work reported in this article was the improvement of quality control in SGML based electronic publishing. In this context we discussed a new SGML authoring and processing model to remedy the lack of semantic validation in the traditional SGML model.

The main idea proposed was to restrict the values that the user could enter, by associating constraints with element definitions. This way we can minimize data incorrectness and improve document quality.

Through the use of several examples we illustrated the main problems in the implementation of such semantic validation task: data normalization, type inference and the definition of a constraint language.

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 this paper we adopted the same solution but we extended it to solve two problems that were raised then: data normalization and type inference. Furthermore we specify in more detail the steps towards the implementation of a constraint language.

Practice as shown us that our approach is a good starting point towards semantic validation implementation. It does not increase significantly the complexity of the system and maintains backwards compatibility with existent SGML applications.

In the moment we are working to transform this prototype into a real application.

11 Acknowledgements

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

We also thank project GeIRA (www.geira.pt) and their sponsors, FCT, INTERREG and FEDER for the case studies used in this work.

References

[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.

[Clark] Clark, James J., *"NSGMLS: an SGML parser conforming to ISO 8879"*, `www.jclark.com`.

[RAH98] Ramalho, J. C., and Almeida, J. J., and Henriques, P. R., *"Algebraic Specification of Documents"*, Theoretical Computer Science, Elsevier, 15 June 1998.

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

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

[RRAH97] Ramalho, J. C., Rocha, J. G., and Almeida, J. J., and Henriques, P. R., *"SGML Documents: Where Does Quality Go?"*, SGML/XML'97 Conference, Washington, USA, Dec.1997.

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

[SB94] Sperberg-McQueen, C. M., and Burnard, L., *Guidelines for Electronic Text Encoding Interchange (TEI P3)*, Chicago, Oxford: ACH/ACL/ALLC, 1994.