Dear reader,

This is a part of my thesis, which explains the way SGML documents are dealt with in an object-oriented system (TBMS). The TBMS is just a description of a system that follows general object-oriented concepts. SGML is a language that is translated to objects and messages in the TBMS.

The section you are looking at only treats the distinction between the structural model underlying SGML, and the exchange model of SGML. These two should in my opinion be separated. The reason this file has been made available on FTP is that I have found may submissions to COMP.TEXT.SGML unclear in this respect. Often the distinction is blurred, which makes the text hard to read and value.

Please let me know what you think.

Arjan

Arjan Loeffen
Faculty of Arts, Dept. of Computing in the Humanities
Achter de Dom 22-24, 3512JP Utrecht, The Netherlands
Phone ++31+30536417, Fax ++31+30539221
E-mail LOEFFEN@LET.RUU.NL

In this section (formatted a bit for printing it as an excerpt) all unresolved cross-references (i.e. to other sections in the thesis) are replaced by (?).

1. SGML as a 'model'

There are two major observations to be made about the SGML standard on documents in general. First, SGML offers a *structural model* for documents irrespective of the way the document is stored or exchanged. Second, SGML defines a language, more precisely an encoding convention, by which documents may be exchanged between systems, which I call the *exchange model* of SGML. Both models are treated in this chapter.
The exchange model is important only for understanding how an SGML encoded document may be mapped onto the DOCUMENT (and related) object type(s). The details of this process and the object responsible for importing the documents (the *parser*) may give insight into both the language and the document model. Current proposals for this mapping do not elaborate on the structure and behavior of the parser used (for example, [ABERE93] and [CHRIS94]). The structural model however is very important as it parallels the way documents are represented as object in the TBMS. It does not depend on the exchange model — the exchange model however does depend on the structural model, in the sense that it reflects the document features through declarations, definitions and encoding strategies applied to the document instance. Below I will elaborate some more on both aspects and their relationships. Please refer to Fig. (?) (pg. (?)) for the separation of structural and exchange aspects of the SGML standard as expressed in the object definitions of the TBMS proposal.

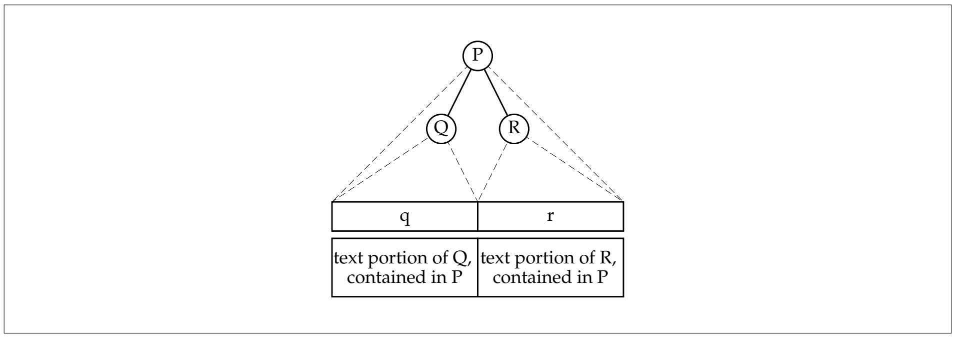

Figure 1 - Elements P, Q and R associated with data portions q and r.

2. Structural model

In SGML, a *document instance* is a collection of 1 or more structural views on textual data. Usually the document instance is defined as a single view, but several interpretations of the same source text may coexist as structurally independent views. These views take the form of a collection of interrelated *elements*, each of which is associated with *data* (Fig. 1). Elements may contain child-elements, that are 'part of' the parent-element. The elements form a tree-structured hierarchy — each element has at most one parent-element. The root of the hierarchy is the *base document element*. The boundaries of elements within one document view cannot cross.

The data associated with any sub-element is by definition also associated with the super-element (*containment*). Within one level of elements gaps may occur (elements do not have to be adjacent), but no gaps are allowed in the data portions associated with an element (see structure shown on the left in Fig. 2). For instance, two non-adjacent citations (Q and R in the figure) may be part of a paragraph (P) element**{! --> ref to TEI: nested quotations and page breaks}**. The running text between the citation data portions (p2) is part of the data portion associated with that paragraph (p1..p3) exclusively. However, the paragraph element is associated with a sequence of characters that has no gaps in it. The structure shown on the right in Fig. 2 can therefore not be modelled (easily) in SGML[1].

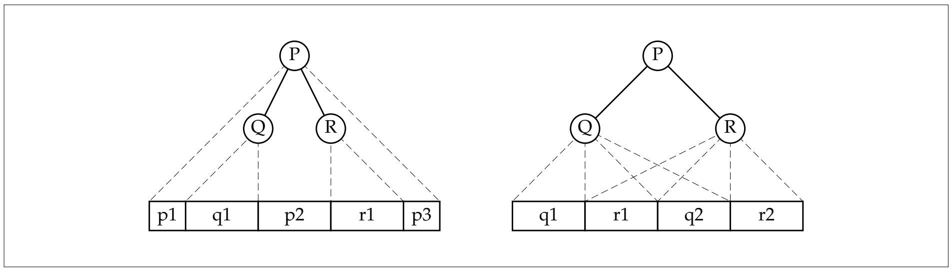

Figure 2 - Elements related to data portions (data). Left relation is correct, right relation is incorrect.

The data may be shared between several concurrent elements in concurrent document structures. For this reason the data is not encapsulated in the element[2].

A document view is of a specific type, called a *document type*. The type of the document view may be described fully by the elements that may occur in it. Therefore, a document type definition is basically a collection of element definitions. However, additional information may be recorded in this definition in order to process the elements correctly.

---

1 If such a relation is required, the most prominent strategy would be define Q to consist of elements QX, that have an ID/IDREF attribute pair for cross-referencing (intended to point to the next element of type QX in the sequence). Note that this strategy is not part of the SGML structural model as, other than by the definition of a content model, the validness of the references cannot be checked by any SGML parser. This point has also been made in note (?) on page (?). The HyTime standard also provides ways to encode such relations. On HyTime, see sect. (?) (pg. (?) ff.).

2 On a discussion of the validness of the concurrent DTD concept, see sect. (?) (pg. (?) ff.).

Elements, in turn, are also typed. Element type names identify:

- the type of the subelements that may occur in it, and in what number and order. This specification may also be 'empty' (i.e. no subelements nor data can occur), be unrestricted (any kind of content is allowed) or be declared to contain data only.
- the attributes of the element, i.e. information on the element that is not part of the text identified[3]. Element attributes may also be used to identify other elements for cross-referencing, special data objects that are stored elsewhere, or even complete documents that have their own, independent structure and specifications. Finally, elements may explicitly identify a program to be used to interpret the data other than by an implied interpretation (the mapping is is called a *notation*). The notation associated with the element text is declared in the document type definition. By default, the notation is that of the system processing the source text, normally a natural language notation. Attributes in SGML therefore have both a descriptive and a structural function.

The type of elements that may occur as part of the parent-element is bound by a model of content, that is associated with the type of the element and the document type it is a part of. This *content model* resembles the right hand side of a context free grammar: it shows what sub-elements are allowed in a specific context, and where data is expected. The content model has both a hierarchical and linear nature. Six operators are available that specify the following linear relations (X and Y are element types):

- X is followed by Y,
- X and Y occur in any order,
- either X or Y occurs,
- X may occur,
- X occurs once or more than once in sequence,
- X may occur zero or more times.

Any group of elements may be abstracted into a separate element (as has been applied rigorously by [CHRIS94][4]). Such a decision is made when the element identifies a complex document substructure that occurs as a subelement of different types of elements, or when such a sequence of elements is a semantic unit, i.e. a specific range of operations only applies to this element.

The content model may also identify the acceptance of data, i.e. running text. The running text is typed as PCDATA, *parsed character data*, which is an exchange concept — it identifies data without any markup left after parsing. A part of the document that has been described as PCDATA may contain subelement and data (through *inclusions*, that are part of the content model). Other primitives that identify the availability of running text at that location in the content model identify the pure textual nature of the element (i.e. the fact than an element is a *terminal element*, i.e. the last element in the element

---

3 For this reason I do not adhere to the strategy outlined by [TOMPA89b], where for performance reasons the element attributes are expanded to complete elements. In that approach the attribute values become the element's content, and cannot be separated from the source text any more.

4 For the reasons explained here, I do not adhere to the approach of [CHRIS94], that does not attach semantics to the class generated, but uses it only as a modelling strategy for content models in terms of (minimal) SETs and LISTs. Minimal model groups are mapped directly onto object attributes, while occurrence indicators are mapped onto object attribute constraints. Exceptions are not mapped.

hierachy). They also outline the way the text is interpreted by a parser. However referenced from the element hierarchy, the source text parsed under PCDATA is 'typed' as *character data*.
Content models thus specify constraints on the structure of the element; all elements (all constraints) form an integral part of the type of the document view.

Notations are processing strategies applicable to the data content associated with an element. Normally the content is written in the 'language' the system understands, but specific markup or other encoding strategies may have been applied in the source text. For instance, graphic encodings in a PostScript image follow the conventions of the language and the procedures defined either as part of the PostScript interpreter configuration file, or imbedded in the image description. This means that the data should be interpreted accordingly. Even a complete SGML document may be seen as encoded according to a notation, in which case the text should be passed to an SGML interpreter in order to process it correctly.
Notations should be seen as specifications of an application that is able to handle some data content. In order to configure the application parameters may be passed to it when the data is to be accessed. These parameter specifications take the form of attribute lists, but are more restricted in domain than element attribute lists. The availability of an *attribute list declaration* on NOTATIONs does not mean that a type relation with SGML elements exist — rather the same construct is used for different purposes[5].

Attributes may be assigned to elements (and notations). They are not part of the running text. They give information on the element associated with some data portion. This information is used by the system processing the document.
Attributes have a 'type', that I will call a *format*[6] (expressed in the *declared value* part of the attribute definition). The format of the attribute restricts the value of the attribute, basically by disallowing certain character sequences (hence the term chosen). In this sense they serve as a template for the value stored, and constitute a first *syntactical* conformance test for a routine expecting a processable string (`"12in"` for 12 inches, `"001"` for an octal number, etc.). Attribute formats may also be declared to test the integrity of the value in the context of the document the element is a part of, and of the system as a whole. In this case the attributes have a structural nature (which is expressed profoundly by the fact that some of these attribute values should not be changed later[7]). Attribute information of this nature may be used for the following purposes:

- identify the element for cross-referencing. In that case the atribute holds a unique name for the element.
- record one or more unique names assigned to elements (by attribute, see previous item) within the document for cross-referencing purposes. These names are intended to be used as the 'target' of the reference; the element defining the attribute is the 'source' of the reference.
- associate a notation with the data portion (see above).

---

5 This issue has been discussed further in [CTS060894].

6 See sect. (?) (pg. (?) ff.) for an explanation of this term in this context.

7 This holds for NOTATION and ID. For this reason it would have been more correct if the standard would separate such attributes from all other attributes, for instance by adding declarations for element identification and reference, and for element notation used.

- identify one/more separate document(s) or data object(s) (an image, a formula) associated with the element[8].

Though attributes are not intended to be used to influence the parsing of the document, the 'structural' attribute types mentioned are checked *semantically* by the parser: notations (e.g. is the notation name declared?), cross-reference identifications (e.g. may all cross-references be resolved uniquely?) and entities (e.g. have the subdocument names been declared?). The registration and integrity maintenance of the attribute values of identifications and notations is also part of the document model, which may even take over this task from the parser. If sub-documents are maintained as separate entities in the system, these values may also be seen as part of the model it supports.
As a result, the structural model of an SGML document distinguishes descriptive attributes from structural attributes on the basis of their declared value. Though structurally separate, they all take the same form in the exchange model, i.e. by attribute declarations and specifications.

Finally, the number of attributes of an element may be enlarged in order to attach additional information to the element found in the tagged document instance. The declaration of this expansion process is called *link process definition* (LPD). There are several types of link processes (simple, implicit and explicit) that have an impact on the structural model. The addition of attributes that are associated with a *linked process* results in a specialization of existing element types. This is explained further in sect. (?) (pg. (?) ff.).

3. Exchange model

When we take a look at an 'SGML document' as it is exchanged, we find a lot of information that does not refer to the structural model treated above. These specifications mostly apply to the way the document is encoded. Again we may outline the basic parts of the model of exchange. This includes some remarks on the way the document is handled by the receiving system.

In SGML, an exchanged document is an *entity* (called a *SGML document entity*), that may take the form of a file, a string, or any other data construct representing a sequence of characters. Entities should primarily be seen as 'character servers' — when required, they pass each next character in the *entity text* to the calling program, until no characters are left. The stream of characters may contain ('imbed') entity references themselves, that are called upon to take over the character emission until no characters are left. When the end of the imbedded entity is reached an *entity end* signal is passed to the calling program, and the character stream of the previous entity is resumed. Ultimately, the document entity passes its own entity-end signal, which completes the parsing process.

---

[8] Though little has been written about what the designers exactly *intended* by allowing *data entities* to be referred to as an attribute value, it seems correct to assume that the entity represents an independent construct (a document, an image, a table) that, in some interactive environment, may be shown in an icon-like fashion to indicate that that object is related to the current element (and possibly all its sub-units). This representation is somewhat similar to the use of margin text or -images in printed matter. Note that the pair of entity types (SUBDOC and NOTATION) that may occur as an attribute value (after parsing) is the same as that which may occur in content. More on this issue in sect. (?) (pg. (?) ff.).

Entities have a *type*[9]. This type refers to a processing strategy known to the parser, i.e. in what way the text should be processed. Normally an instance will be processed as a stream of parsable character data without any special notifications (the replacement text is entered as a *parameter literal* only). However, an entity may be defined to contain a special kind of characters (system characters or character data — the characters are not parsed but inserted as-is), or to be treated as a predefined markup construct (a tag, a marked section, or a processing instruction). In the latter case, implied markup constructs are added to the replacement text, and appropriate actions are taken. Next, entities may be internal or external to the document set. External entities may follow an interpretation convention, ie. a *notation*. When a notation is applicable, there are two additional types of information to be recorded in the external entity declaration: how to go about reading the data into memory, and what processor to use to interpret the encodings stored there. The character sequence may be character data, non-SGML data or system data. The notation applicable is recorded next.
As an alternative to declaring entities to be handled by some external processor, the entity may be processable by the SGML processor *itself*. That case the entity holds a *subdocument*. (Sub)documents have their own document type definition, which sets them apart from all other entities. The (sub)document entity is in fact a reference to an independent parsing process, of which the results (the document and dependent datastructures created) are made part of the calling parsing process[10]. The document entity is one in a set of entities known to the parser. The associations between an entity name and the data construct holding the entity text (called the *replacement text*) is local to the document view. The replacement text may be local to the document view, the system, or a range of systems. In the latter case, the entities have a name that is unique over a range of systems, and may therefore be subject to standardization. Entities may be added to the set of entities available to the parser by an *entity declaration*, in which case a new association between an entity name and the replacement text is added to the entity set of the document view.

In short, every character sequence parsed is part of an entity. Entities therefore are not part of the document model — they are exchange constructs. However, the document may expect some replacement texts to remain available in some form when the parser has completed its task. This could apply to character entities, i.e. representations of characters that are not part of the character set of the system, and external entities that remain a single, but parsed, unit and are referenced from within the parsed and stored document.

The characters supplied by the active entity (the replacement text) may either be *markup* or *character data*. When a character is not markup, it may be part of the declared character set of the document instance (*SGML character*) or foreign to that character set (*non-SGML character*). If a character (or character sequence) belongs to the document character set, the decision which *function* the character has in the document instance is made by checking the context in which the character appears. The *context* is all information gathered from previous and following markup. This information encompasses:

- the declared function of a character (can the character start a *name*?)

---

[9] The type of the entity (§ 109) cannot indiscriminately be mapped onto a 'type' in the sense of the TBMS, as most entities are resolved by the parser. More on this in sect. 4 (pg. 11 ff.).

[10] For clarity purpose I will refer to the 'parsing process' by the word *parser* (i.e. the minimal environment for the recognition of features of a single document); in line with the SGML standard, the encoded document is called the *document instance*.

- the location within a markup construct (is any *tag* currently open?)
- the minimization strategies valid for the current element (is the character part of a *data tag* defined for the current element?)
- the type of text or notation applicable to the current element (are general entity references allowed here?)
- and so on.

The parser keeps track of the context by saving the *recognition mode* that is started by a parsing event, and restoring the previous recognition mode when the event is finished[11]. Each recognition mode is in fact associated with a range of possible characters, constraints on the recognition of these characters, and a parser action to be performed when a character with a recognized *role* is encountered. For any document instance the parser works identically, allthough:

- character functions, content models, minimization strategies and so on may differ.
- the set of recognized constructs may be enlarged.

Therefore, the parser is defined on an abstract level; a special section —the *SGML declarations*— is read first in order to assign true characters and functionality to the parser (more on this in sect. (?)). In the broadest sense, all characters are part of the document markup — each character plays a role in the markup recognition process. In a more restricted sense, markup takes the form of declarations, processing instructions, tags and entity references:

- *Declarations* mostly tell the parser what kind of constructs are used in encoding the document instance. This concerns the declaration of document types (types of document views), element types and attributes, notations, entities, and link type definitions. By nature, these declarations are parsed and mapped onto system constructs before the actual document instance is read. Declarations are also used to regulate the parsing process. Data tag mappings are declared which tells the parser to use a different recognition strategy. Marked section declarations tell the parser to skip or include some content, or treat it as data. All these constructs are solved and/or executed by the parser, and are not reflected in the structural model.
- *Tags* tell the parser that an instance of an element is started or is complete. The *start-tag* is used to specify the type of the element and the actual values for attributes that are defined earlier on the element type. The parser checks if required attributes are set, or inserts implied, fixed or current attribute values. The character sequence between the start-tag and the end-tag is passed to the application (e.g. the object representing the element), unless a new element (i.e. start-tag) is encountered. In this respect, *minimizations* are parser constructs, that allow for impliable start- or end-tags in certain contexts. Tag- and minimization information is not reflected in the structural model — they are vehicle for representing the structure of the document when the document is exchanged.
- *Entity references* are resolved as explained above. Entity references are parser constructs. Their impact on the structural model depends on how we interpret the reference. We may assume that if the reference represents an entity that may be parsed by the SGML processor, the replacement text is inserted *and the connection with the original entity is lost*. Changing that entity will have no impact on the replacement text inserted. If the reference is external (a notation entity, or a subdocument), the reference is retained in some form, and a modification to the external entity is reflected in all documents that refer to it. Note that this approach

---

11 The construction of the SGML parser is not part of [ISO8879].

is not part of the standard; some system may copy the external entity (e.g. the PCX image, the formula) to its data store and break the link with the external entity.

- *Processing instructions* explicitly invoke some action by the system (and are therefore system-dependent). They are not part of the structural model, but are, other than the prior three markup constructs, passed on to the application that reads a canonical form of the exchanged document (as treated below).

What should have become clear from the above is that a large part of the exchange model of SGML is relevant only for the human reader of the encoded document, or, as R. Mossis states in [TEI260593], *"The idea that humans, instead of software, should be reading the markup is probably a perversion of SGML's historical intent and in any case a terrible idea"*. On the other hand, E. Naggum reacts on [SGML080894] to the same assumption (made by me on [CTS]): *"the WhirlWind Guide only makes it clear that we have at least thrice as far to go as we have come to even get close to some of the "ASCII editing environments" in power and utility, and they will leap forward while those SGML tools snail forward to catch up"*. It is nevertheless my opinion that, even if such tools are not widespread for reasons of early development or marketing, they will be available and will make many constructs in the SGML exchange model, such as data tags, short references and minimization, redundant. Undesirable implications of using short references are treated in [HERWI94:179ff].

```
AIDENTIF TOKEN OIAL2
AXREF IMPLIED
ASTOCK TOKEN 67110
ASTATUS TOKEN SUBMIT
(LETTER
(GENERAL
(HEADLINE
(HP3
-once in a lifetime:
)HP3
- a new way of \nsaving your money
)HEADLINE
(FOOTLINE
(HP3
-global insurance ltd. 2 newark lane,
\nglasgow.
)HP3
)FOOTLINE
)GENERAL
(HEAD
(ADDR
(ADDR.NAME
-Mr. J. Hughes
)ADDR.NAME
(ADDR.STREET
-Fleetstreet 17
)ADDR.STREET
(ADDR.TOWN
-London, England
)ADDR.TOWN
)ADDR
(POSTMARK
(POSTMARK.TOWN
-Glasgow
)POSTMARK.TOWN
(POSTMARK.DATE
-march 26, 1992
)POSTMARK.DATE
)POSTMARK
(RELNR
-0651166
)RELNR
)HEAD
(BODY
(DEAR
(SIGNON
-Dear
)SIGNON
(SURNAME
-Mr. Hughes
)SURNAME
)DEAR
(P
-You too can keep your money in our new
bank! This bank\nis a subdivision of our
insurance company
(HP2
-Global\nInsurance
)HP2
-, and is therefore much more reliable
than\nmost other banks.
)P
(P
```

```
<(LETTER)LETTER STATUS="SUBMIT"
STOCK="67110"
IDENTIF="OIAL2"><(LETTER)GENERAL><(LETTER)HE
once in a lifetime:
</(LETTER)HP3>
 a new way of
saving your money

</(LETTER)HEADLINE><(LETTER)FOOTLINE><(LETTE
global insurance ltd. 2 newark lane,
glasgow.

</(LETTER)HP3></(LETTER)FOOTLINE></(LETTER)G
Mr. J. Hughes
</(LETTER)ADDR.NAME><(LETTER)ADDR.STREET>
Fleetstreet 17
</(LETTER)ADDR.STREET><(LETTER)ADDR.TOWN>
London, England

</(LETTER)ADDR.TOWN></(LETTER)ADDR><(LETTER)
Glasgow

</(LETTER)POSTMARK.TOWN><(LETTER)POSTMARK.DA
march 26, 1992

</(LETTER)POSTMARK.DATE></(LETTER)POSTMARK><
0651166

</(LETTER)RELNR></(LETTER)HEAD><(LETTER)BODY
Dear
</(LETTER)SIGNON><(LETTER)SURNAME>
Mr. Hughes
</(LETTER)SURNAME></(LETTER)DEAR><(LETTER)P>
You too can keep your money in our new
bank! This bank
is a subdivision of our insurance company
<(LETTER)HP2>
Global
Insurance
</(LETTER)HP2>
, and is therefore much more reliable than
most other banks.
</(LETTER)P><(LETTER)P>
The most important features of banking in
the new 'once
in a lifetime' way are
<(LETTER)LIST
BULLET="DOT"><(LETTER)ITEM><(LETTER)P>
One single deposit for the period of 30
years.

</(LETTER)P></(LETTER)ITEM><(LETTER)ITEM><(I
Interest rate of 20% on a yearly basis
<(LETTER)FN><(LETTER)P>
To be paid at end of term.

</(LETTER)P></(LETTER)FN></(LETTER)P></(LETT
Safekeeping of your well-earned money

</(LETTER)P></(LETTER)ITEM></(LETTER)LIST></
We are sure that you will be interested in
our offer.
Please contact your local insurance dealer
```


Figure 3 - Two output formats of parsers applied to the `Letter` instance, shown in Fig. (?) (pg. (?)). On the left SGMLS (a native format), on the right the canonical form, generated by MARK-IT.

4. Connection between structural and exchange model

The exchange model is the result of the need to exchange documents that are built according to the structural model of SGML. A document encoded in SGML may be read by an application and translated into its own data structures in order to process the document further (edit, query, print, etc.). Though a structural model may easily be extracted from the standard for SGML, the way the encoded document is represented internally is *not* part of the standard. In this chapter it is however assumed that the structural characteristics of the document should be preserved, while the exchange characteristics may be ignored.

Generally an SGML conforming application uses a separate parser included to parse the document exchanged into a much simpler encoding. The result of the parse is some form that throws out all exchange aspects of the document, that we may call the 'local exchange form'. If the local exchange form may be parsed again by an SGML parser, it is called a *canonical* (or *complete*) *form* ([HERWI90:169], see Fig. 3, page 10, on the right). However, any form that is understandable by the application may be generated — for instance, the SGMLS parser generates a file in the format shown in Fig. 3, on the left.
Some parsers have an application programming interface (API) that allows the application to streamline the connection between the application and its parser. Such an API would be used to insert information in the parser's output file that is needed to correctly interpreted the local exchange form.

As stated, the result of parsing a document instance (whether canonical or not) should reduce the encoded document to a file in which all matters of exchange are resolved to basic and explicit information packages. To be more precise, the following translations are carried through:

- All omitted or shortened tags are inserted in full. Data tags and short references are mapped onto tags. Ranked element GI's are suffixed with the number implied.
- Marked sections are left out or inserted as required.
- Processing instructions are retained in full.
- All replacement text for general and character entities is inserted at the appropriate location. External entities are resolved by either inserting the replacement text or by specifying the location of the entity according to the (inferred) system identifier.
- Subdocuments inserted are identified by a start-code and an end-code. This ensures that the application is capable of separating the elements of both documents.
- Link type specifications are resolved by assigning linked attributes to elements.
- Character data in content is output in such a way that ambiguities do not occur when reading the generated text.

The last item in this list applies to the canonical form in particular. The canonical form should be re-parsable by an SGML parser that is unaware of those exchange aspects that have been resolved in the canonical form. This means that the parser does not have to understand the mechanisms involved in generating a canonical output, i.e. comply with the items listed above.

The result of such a 'pre-parse' is a document with:

- element start- and endsignals. In the canonical form they take the form `<(A)B ..attspecs..>`, identifying the name of the *document type* to which the element belongs (a *name*, A)[12], the *element type* (or *generic identifier*, B), and possibly an *attribute specification list*. End tags take the form </(A)B>. Subdocuments included as general entity references are included in full, or are identified by some unambigous reference.

- character data sequences. Characters belonging to the running text covered by the element. General entity text is inserted, character entities are resolved.
- external entity references, with, if applicable, information about the notation used.
- processing instructions. Instructions to be handled by the processing system.

Any SGML parser can translate the exchanged document to a document in such a local exchange form. Therefore, in the treatment of the document type in sect. (?) (pg. (?) ff.) it is assumed that such a form is available.

In this thesis, the local exchange form generated by the SGMLS parser (a conforming SGML validating parser[13]) is used, mainly because this form is easy to read. However, where aspects of the standard not supported by SGMLS are treated, the canonical form generated by the conforming SGML validating parser[14] MARK-IT is used in stead.
Note that the choice of local exchange format is irrelevant in this context, as these formats may be converted into each other assuming the parsers have an identical conformance level (on conformance, see page (?)).

5. Restrictions in the model

As may have become clear in sect. 2 (pg. 2 ff.), SGML does not offer a text model in the sense of sect. (?) (pg. (?) ff.). A *text model* not only conveys the text structure, but also the operations that are allowed and relevant for text handling, and constraints that apply on both structure and operations. This concerns both the physical and the logical side of the system. SGML only describes the text structure, and constraints on structural transformations. Structure-dependent operations may be defined, but these are outside the scope of the standard. Below structure and constraints are treated shortly, in line with sect. (?).

Physical structure. SGML focusses on physical aspects of the documents in as far as the documents portability and readability is concerned. The physical characteristics are globally defined in the *SGML declarations*. This first section of the SGML document defines the characters used and their individual roles, the memory requirements, the availability and form of the tags in context, and the form of delimiters and names. These declarations make the physical structure of the document platform-independent. A second section, the *document type definition* (DTD), also records physical characteristics of

---

12 Concurrency is not supported by SGMLS, therefore no start-codes for such elements are defined. A future version however could introduce concurrency information by extending the GI's with the *document type specification*. The same holds for unresolved entities (eg. external entities).

13 SGMLS is a public domain parser generator based on ARCSGML, originally designed by C. F. Goldfarb, written by James Clark. SGMLS supports the features SHORTTAG and OMITTAG, SUBDOC (infinite level), and FORMAL. All other features are not supported.

14 MARK-IT is a product of Sema group, Belgium, [SEMA90]. MARK-IT supports all features. It produces a canonical form, but the output may be mixed with special statements in the system language using the API.

the document, as explained above. All such declarations tell the parser how to read the physical document. However, it should be stressed that a *conforming SGML system* (page (?)) is not forced to use such SGML encoded documents as the local text format. The declarations apply to the document as it is *exchanged*.

Logical structure. The logical structure of the document as defined in the DTD is identified by the *document type declaration*. The DTD describes what elements may occur at a specific location, and which attributes they (may) have.
The logical structure of the document reflects the complex relations. These may be:

- *Hierarchical* (elements consisting of subelements, implementing *containment*)
- *Sequential* (elements following other elements)
- *Optional* (elements that may occur at a specific location)
- *Imbedded* (elements that may or may not occur anywhere in a contained element)
- *Referencing* (elements that are identified from a specific location)

The elements are typed, implying an abstract form (*content model*) and additional information (*attributes*). Types assign an information value to the individual elements, e.g. *factual* information on ownership, creation date, location of original; *type* information, such as indication of paragraph, item, title; general *form related* information, such as the language used, character encoding scheme, and the like; *critical* information, such as readability of original section(s), emendations; *layout* information, such as fonts, typefaces, indentation and page feed; *content* information, such as metaphor, surname, citation, et cetera.

Physical constraints on structure. Although the SGML declarations may allow for much freedom in encoding the document, all character and feature use is restricted by these declarations. For instance, if *tab* (dec 9 under ISO 646) is not defined as a function character, entity references that use *tab* as a *reference end* will not be resolved (correctly). Therefore, such constraints are implicit to the SGML declarations. As an other example, the QUANTITY clause defines some constraints on nesting levels for entities, model groups and tags. The way the document is represented internally is not constrained by SGML in any way.

Logical constraints on structure. The use of content models for elements is the heart of the content validation of SGML. These content models specify exactly where and how a type of element or a data portion may occur in the document instance. As a result, the *document type* is a *structural* concept in SGML. It may even be argued that SGML is a language for defining structural constraints for documents. However, SGML does not model the *semantics* of the contents of the documents (neither represented by element structures nor by element attributes). Constraints on this level cannot be formulated, and remain under control of the application. As attributes may be used as a means to define structural relationships or features, constraints may defined on attribute declarations and -values.

ABERE93 -- <Aberer et al, 1993> K. Aberer et al: 'The prospect of publishing using advanced database concepts'. In: **Electronic Publishing** 6 (4) 1993. Pp. 469-480.

CHRIS94 -- <Christophides et al 1994> V. Christophides et al: 'From structured documents to novel query facilities'. In: **Proceedings of the ACM SIGMOD conference on management of data.** 1994. Pp. ???? (Also available from ftp.inria.fr /INRIA /Projects /Verso)

CTS# -- <comp.text.sgml> USENET discussion group **comp.text.sgml**. Since sept. 1990. (Also available in edited form as mailing list comp-text-sgml@naggum.no).

HERWI90 -- <Van Herwijnen 1990> E. van Herwijnen: **Practical SGML**. Dordrecht [etc.], 1990 (Kluwer academic publishers).

HERWI94 -- <Van Herwijnen 1994> E. van Herwijnen: **Practical SGML**. Boston [etc.], 1994 (2nd ed., Kluwer academic publishers).

ISO8879 -- <ISO 8879-1986/1988 (E)> Information processing - Text and office systems - Standard generalized markup language (SGML). 1986, with amendment 1, 1988 **(ISO 8879-1986/A1:1988 (E)).**

SGML# -- <SGML-L> SGML-L SGML-L mailing list. SGML-L@DHDURZ1.BITNET

TEI# -- <TEI-L> TEI-L Text Encoding Initiative public discussion list. TEI-L@UICVM.BITNET

TOMPA89b -- <Tompa 1989b> F. W. Tompa: 'What is (tagged) text?'. In: **Dictionaries in the Electronic Age.** (Proceedings of the 5th conference of the University of Waterloo Centre for the New OED) Sept. 1989. Pp. 81-93