Automatic generation of SGML content models *

Helena Ahonen
Department of Computer Science
University of Helsinki
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
E-mail: helena.ahonen@helsinki.fi

Abstract

We study the problem of automatic generation of a document type definition (DTD) for a set of Standard Generalized Markup Language (SGML) documents. We present various situations where we have tagged documents but no DTD, and discuss the requirements various applications may have with respect to the generation process. We also present an automatic DTD generation tool that can be adjusted for several tasks necessary in the applications. The method is also demonstrated with some experimental cases.

1 Introduction

An SGML (Standard Generalized Markup Language) [1] document has both *tagging*, i.e. all its elements are marked with begin and end tags, and a *document type definition (DTD)* that describes the allowed structure of the document class in which the document belongs. The basic components of a DTD are *element declarations* that contain for each element a *content model* which in turn specifies the *model group* that defines the allowed content of the element. A model group is constructed of simpler model groups, sequence connectors, and occurence indicators.

There are, however, situations, where we have tagged documents, but no DTD is available. This may happen, for instance, when we are converting documents from non-SGML environments, e.g. as a part of a large design process or with quick online transformations. Clearly, there is a need for *automatic DTD generation*.

Automatic DTD generation has also been studied in [2, 3, 4, 5], mostly in connection with some specific situations. We try to give an overview of the various cases where the need for a DTD may arise, and we also present our own method that has features adjustable for these cases.

Assume we want to find a DTD for the following tagged dictionary entries [6].

```
<Entry><Headword>kaame/a</Headword> <Inflection>15</Inflection>
<Sense> kammottava, kamala, kauhea, karmea, hirvea, pelottava </Sense>.
<Example_block> <Example>Kaamea onnettomuus, verityo. </Example>
<Example> Tuliaseet tekivat kaameaa jalkea. </Example>
<Example> Kertoa kaameita kummitusjuttuja. </Example> </Example_block>
<Sense_structure> <Technical_field>Ark.</Technical_field>
<Example_block><Example>Kaamea hattu.</Example><Example> On kaamean
kylma. </Example> </Example_block> </Sense_structure> </Entry>
```

*This work was partially supported by the Academy of Finland and TEKES.

```
<Entry><Headword>kaameasti</Headword><Example_block>
<Example>Sireenit ulvoivat kaameasti.</Example> </Example_block></Entry>

<Entry><Headword>kaameus</Headword><Inflection>40</Inflection>
<Example_block> <Example> Sodan kaameus.</Example></Example_block></Entry>
```

There are at least three trivial solutions:

1. Find each element within the document, and allow it any content:

```
<!ELEMENT Entry                ANY >
<!ELEMENT Headword             ANY >
<!ELEMENT Inflection           ANY >
<!ELEMENT Sense                ANY >
<!ELEMENT Example_block        ANY >
<!ELEMENT Example              ANY >
<!ELEMENT Technical_field      ANY >
<!ELEMENT Sense_structure      ANY >
```

2. For each element find all the elements that appear within it, and form an optional and repeatable model group:

```
<!ELEMENT Entry                (Headword | Inflection | Sense
                                | Example_block | Sense_structure)*>
<!ELEMENT Sense_structure      (Technical_field | Example_block)*>
<!ELEMENT Example_block        (Example)*>
<!ELEMENT Headword             #PCDATA  >
<!ELEMENT Inflection           #PCDATA  >
...
```

3. De-facto grammar; take all the structures of the instances to be the DTD:

```
<!ELEMENT Entry            ((Headword, Inflection, Sense, Example_block,
                           Sense_structure) | (Headword, Example_block)
                           | (Headword, Inflection, Example_block)) >

<!ELEMENT Sense_structure  (Technical_field, Example_block) >
<!ELEMENT Example_block    ((Example, Example, Example)
                            | (Example, Example) | Example) >
```

Solutions 1 and 2 are usually overgeneralizing, while solution 3 is too restrictive. Non-trivial solution should find a good compromise between these two extremes and capture at least some knowledge of the order of the elements, and whether the elements are optional, required, or iterating, like in the following:

```
<!ELEMENT Entry            (Headword, Inflection?, Sense?,
                            Example_block, Sense_structure?) >
<!ELEMENT Sense_structure  (Technical_field, Example_block)  >
<!ELEMENT Example_block    (Example)* >
```

Of course, we can ask whether we need non-trivial DTDs in the first place, since tagged documents may be used even without any DTD. For instance, we can make structured queries that just specify a string inside some element. However, it is certainly difficult for a user to get the most benefit from the structure, if s/he does not even know it. Many application programs, like editors and transformation tools, require a DTD. A trivial DTD may not be satisfying, if we, e.g., have strong validation needs. In the next section we discuss further the ways in which we can utilize automatical DTD generation, and in Section 3 we present the DTD generation tool we have implemented. Section 4 shows the results of some experiments that illustrate the use of our method in some application areas.

2 Applications of automatic DTD generation

Automatic DTD generation can be used for many purposes. The actual needs of the application determine how the generation has to be applied, and what kind of result we can expect. The needs can be examined as the following dimensions:

Complexity of the structure. If the structure of the document intances is simple, automatic DTD generation is easy and the quality of the result is good, i.e., we can easily achieve all the requirements stated below. Hence, in the following, we assume that the structure is somehow complex.

Time frame. We may have online application that requires immediate result. Sometimes we need the result quickly, and sometimes. e.g., in a design process, we have more time.

Need of completeness. The result either has to be a valid SGML DTD that is immediately usable with an SGML application, or the result is used as a source of information and hence may be a partial solution.

Need of structurality. Some applications just need some DTD, thus a trivial solution that loses most of the structure may be appropriate. If a trivial DTD is not adequate, there is usually some tendency how much trivialization is preferred. If a DTD generation is used mainly to gather some information of the documents, the more structure can be discovered the better.

Need of readability. The resulting DTD may be used directly by some application program, hence the machine-readable form is adequate. Instead, if the DTD has to be understandable also by human users there are different requirements for readability.

Tagging can/cannot be modified. The source instances may be "given as such", i.e., there is either a fixed structure, or there is no time to modify the structure. On the other hand, for instance in a design process, the DTD generation can be used to reveal problems with the instance structures and tagging, which may be then updated according to the results.

Batch or interactive processing. Usually there is always first some kind of batch processing, which is followed by an interactive session if necessary, and if there is time and people for that.

In addition to the interactions already mentioned, these dimensions affect each other in several ways. The most important of these is: If the DTD has to be complete, and the structure is complex, we cannot get both full structurality and readability, i.e., there has to be some trivialization. This is even emphasized, if the process has to be online, whereupon the user cannot simplify the result interactively.

There are at least three cases where the other dimensions do not affect the result, in addition to the first case mentioned, i.e., when the structure is simple. Clearly, if some trivial solution is adequate, we can always form it quickly, it is complete and readable, and there is no need to change the structure of instances. Similarly, if we do not require completeness, a somehow partial solution should be easy to generate. The fourth case is if a machine-readable form is satisfactory.

2.1 DTDs for static documents

Some document collections are "given as such", e.g. archives may have a very conventional structure, and it is not possible to modify it. Hence there is no need for a design process in the conventional sense, but the task is to find a DTD that makes it possible to use the documentation in the best possible way with SGML application programs. Moreover, with this kind of "historical" documents, which were never designed any grammar in mind, it may be very interesting to see the exact structure found in the documents. Hence, the discovery of the structure both has a meaning as such and gives extra information for various processing needs.

The second case of static documents are documents that come from some external source, e.g., messages to our company from some other companies. Now we may want to transform these messages to our own format, and for the transformation we need some DTD for the source documents.

2.2 Interactive design tool

Above we considered static documents. Quite in the contrary is the situation when we are designing our own document collection, e.g., all the documentation needed in our enterprise. In this case DTD creation is more or less similar to database design: modelling of the application area is needed. Now automatic DTD generation can be used as one design tool. If the planned documentation is based on existing documents, we can first gather information from their structures and use it for evaluation and further design.

There might be a situation that the documentation was done e.g. with MS-Word styles, and the writers were obliged to use the styles in a rigid way. When we then want to convert the whole documentation to SGML, we might hope to get the DTD directly from these style rules. However, people do not necessarily obey the rules very strictly, since there is nothing in Word that could make them obligatory. When the documents are converted into SGML, it is useful to check what actually is the real structure. All the deviations of the rules might not be totally negative: there might be ways the people have created to overcome some in the design process unexpected situations. Hence, the automatic DTD generation can reveal useful knowledge that can benefit the design of the new SGML documentation.

2.3 DTDs for different views and subdocuments

It is often wise to use some standard DTD. However, these DTDs are often very large and designed for many varying cases. If we then need a simpler DTD for some task or a DTD for some subdocument, we can use automatic generation to find a DTD that accepts the selected documents. One example of this kind of DTD is an author DTD [7]: a DTD that is given to the people that write documents. For instance, in our research project, we have used ISO 12083 standard DTD for books [8] as a DTD for engineering text books. First, the authors used MS-Word to write the original text. However, the conversion process is very tedious, and also the writers find it better to start to use some SGML extension to Word. Therefore, we need now a simpler DTD that the writers can use. We can use the already converted tagged textbooks to generate a DTD automatically, and check the result against the large DTD to make the obvious generalizations, like allowing more than one author.

2.4 Online DTDs

We have studied in our research project *intelligent assembly* of documents, which means that the user can configure new, individualized documents from a collection of documents and possibly also from external information sources. In document assembly, the basic operations are *queries* that return the selected fragments of documents, and *configuration* of these fragments to form sensible documents. Hence, the assembled document is a transformation of one or more existing documents. If, instead of simple printing of documents, we want that the resulting documents are valid SGML documents that can be reused later, possibly in assembling new documents, we need a DTD for them. These *online DTDs* have to be generated automatically. Of course, we can always use some trivial DTD, but if we want to get the maximal use of the assembled document, it is better to find a solution that is more structured.

We can form a DTD from the tagging of the assembled document, but this DTD may be too restricted. We are going to study, how the source DTD and the transformations can be used to generalize the target DTD in a suitable way.

3 Automatic DTD generation

We have designed and implemented an automatic DTD generation tool [9, 10] that is adjustable according to the above mentioned dimensions. In a typical session, there is a batch processing phase (Section 3.1) that produces the first complete candidate for a content model. The content model is a generalization of the source instances, but usually highly structural. If the result is readable enough for the purpose of the application, there is no need for further processing. Otherwise, the continuation depends on the situation.

If we have an online application but still need a readable result, there has to be some further automatic generalization. This can be done by dividing the problem, e.g. with *isolating* some model groups (Section 3.2), or by *trivializating* the result (Section 3.4). These further generalizations can be started every time after the basic part, or there may be some way to measure the complexity of the result, whereupon exceeding some given complexity threshold triggers the operations.

The above generalizations can be used even if we can afford interactive processing, although they can be applied more intelligently, since the background knowledge of the user can be utilized. For instance we can try to *discover inclusions* automatically (Section 3.3), but the result is better if the user can evaluate the candidates found and select the actual inclusions among them. If the automatic generation is used as a design tool, a useful operation is the *separation of common structures* from exceptional ones (Section 3.5) that generates a content model for both cases. This can give valuable information for the design process and may also cause some updates in the instances.

3.1 Generating unambiguous content models

The basic idea of our method is to form for each element a deterministic finite automaton from the structures of the instances of this element, generalize these automata in a certain way, and then convert the automata to regular expressions that are easily written as SGML content models. The content models should be, according to the SGML standard, *unambiguous* in the following sense: A content model is ambiguous if an element or character string occurring in the document instance can satisfy more than one primitive token in the content model without look-ahead. The unambiguity of content models has been studied in [11], and we have used its algorithm as a basis to generate unambiguous content models [10].

3.2 Isolation of model groups

If the document structure is complicated, it is often useful to be able to introduce more nesting in the structure than present in the instances. For example in the dictionary entries seen in Figure 2 there are clearly elements that form the first part of the entry (e.g. Headword (H), Inflection (I), Consonant gradation (CG)) and others that appear in the end (e.g. Sense (S), Example (E), Technical field (TF)). Since both of these parts vary a lot, processing them separately reduces the complexity of the content models remarkably. The separation can be done by *isolating a set of elements* to form a new model group.

There are two ways in which the isolation of elements can be used in the structuring process. First, the new model group can be described as an entity that can then be included in various other model groups as a kind of short-hand notation (see an example in Section 4.2).

Using of entities does not affect the document instances, but if elements are formed instead of entities, also the instances have to be updated. Then the isolation should generate a parser that can convert original documents to contain the new structures. The second way could be useful, if the isolation reveals that some essential structure is missing in the original documents.

In the previous we assumed that the user is aware how the parts usually appear in relation to each other. Sometimes, however, this knowledge, or even the user, is not available. Therefore, it is useful, if there is some way to discover the strongly related parts automatically. In our method we apply clustering.

Definition 1 Let A and B be elements.

- n_{AB}: how many times A appears on the left side of B. If B never appears next to A, then n_{AB} is undefined.
- n_A: how many times A appears in the document.

Now we can compute distances for a pair of elements, and, further, distances between sets of elements, i.e., candidates for clusters. After finding the best clusters the method can either proceed automatically, and isolate the respective elements, or the clusters can first be presented to the user for evaluation. Similarly, the method can either create the names for the new entities or elements itself, or it can ask them from the user. Hence, the automatic isolation can be applied both in batch and interactive processing.

3.3 Inclusions

If some element appears more than once with many other elements, this element may be floating. Then it could be reasonable to interpret this element as an inclusion, since undiscovered inclusions may reduce the readilibity of the resulting content model remarkably. Examples of floating elements include footnotes, figures, and various cross-references.

To find the candidates for inclusion we can use the same measures n_{AB} and n_A as in automatic isolation. We simply count the number of elements that appear next to an element A, and if the total count is greater than a given proportion of the whole number of elements, then A is considered an inclusion. After a possible check by the user, the inclusion element has to be removed from the content and inserted to the set of inclusions.

3.4 Local trivialization

The whole generalization process attempts to find a solution somewhere in between the exact set of instances and the trivial case that contains all the structures. Basic generalization obtains some solution, but if it still contains too much variation, it can be further generalized. *Local trivialization* attempts to find the most complicated substructures, and starts to generalize these. If the complexity of some model group exceeds a given threshold, we form an optional and repeatable model group of all the elements included. For instance, in a textbook a paragraph may contain several other elements, like formulas and lists, in varying combinations. In the following, the part `(DFORMULA | LIST | EMPH | FORMULA | SUBSCR | SUPERSCR | FIGREF)*` is a locally trivialized part.

```
<!ELEMENT P        - -   ( #PCDATA | (DFORMULA | LIST | EMPH | FORMULA |
                                SUBSCR | SUPERSCR | FIGREF)*
                                (FIGGRP, FIGREF? | SECREF | SYMBOL)?
                                | FIGGRP | FORMREF | SECREF |
                                (SYMBOL, SYMBOL?)  | TABLE ) >
```

3.5 Common and exceptional cases

Frequency information, i.e. how many times a certain structure appears in the instances, can be used for quantifying the importance of different types of structures for the element. At the moment our method produces one content model for the common cases and one for exceptions, according to a given threshold. This knowledge can be used, e.g., for discovering errors and rare cases in the structures, which may lead to modifications in tagging, if that is allowed.

4 Experimental results

We have implemented the method described above, excluding the discovery of inclusions. In this section we present some experiments that illustrate how our method can be used to satisfy needs of varying applications.

4.1 Textbook

As mentioned earlier, we have converted one textbook into SGML and structured it according to our ISO 12083 -based DTD [8]. As we need a simpler DTD for the authors of new books, we generated a DTD for the book we already have, and now we can use this DTD as a basis for the author DTD. In Figure 1 we can see all the non-trivial element declarations resulted, i.e., the content models for all the elements seen on the right-hand side but not on the left-hand side are #PCDATA.

Our method produces a complete DTD, although in this case it is not totally necessary. After the basic generalization phase the method continued with local trivialization. It trivialized some model groups that have large variation, e.g. a part of the declaration for paragraph (P). All the processing was done in batch mode.

4.2 Dictionary

Most challenging of our test cases has been the part $A - K$ of a Finnish dictionary [6]. We converted the typographical tags of the dictionary, which consists of about 16000 entries, to structural tags, and obtained a set of 468 distinct structuress. Every structure also received a frequency, i.e., the number of entries that the structure covers. We chose 55 of the most common structures (Figure 2), which together covered 14791 entries.

The dictionary was not originally designed for computer use, and therefore the structures of the entries have great variation. Even the editors cannot specify the desired structure for an entry. Hence, there is a strong need to gather information from the existing structures and use this knowledge to update instances to develop a more consistent structure. In this point this case differs from some otherwise similar cases, e.g., discovering the structure of ancient books or archives, where it is not desirable to change the structure.

Again, after the basic generalization with local trivialization the result was the following:

```
<!ELEMENT EN    - -    (H, ((EX | TF | S)*, R? | I, ( (EX | TF | S)*, R?
                        | CG, ((EX | TF | S)*, R? | (R, EX?)  | PrF | (BW, EX)
                        | PaF, ((EX | TF | S)* R? )? )?  | (R, EX?)
                        | PI, (S | ((TF | EX | S)*, R?) | R )?
                        | PaF, ((EX | TF | S)*, R? )? | PrF
                        | II, ((TF | EX | S)*, R? )?  | (BW, EX))? | (R, EX?)
                        | PrF | PI, (S | (TF | EX | S)*, R? | R )? )?) >
```

This time we used some backgroung knowledge, i.e., that elements H, I, CG, PI, and II form the first part of each entry. Therefore we made one interactive isolation step and obtained the following declaration:

```
<!ELEMENT EN    - -    ( %HP, (((EX | TF | S)*, R?)  | (R, EX?) | PrF | (BW, EX)
                        | (PaF, (EX | TF | S)*, R? )?) )? ) >

<!ENTITY % HP          "(H, (I, (CG | PI | II)? | PI)? )" >
```

5 Conclusion

We have discussed the possible uses of automatic DTD generation tools, and which characteristics are needed from such tools in the applications. We introduced seven dimensions that affect the process of DTD generation: complexity of the structure, the time frame, the need of completeness, the need of readability, the need of structurality, the possibility to modify the tagging, and the possibility to utilize interactive processing. Every application sets its own values for each of these dimensions, and

```
<!ELEMENT ANSWER        - -   (TITLE?, P, ( (P, FIGGRP? | FIGGRP)
                               (P, FIGGRP?)*, FIGGRP? )? ) >
<!ELEMENT APPENDIX      - -   (NO, TITLE, (P | (SECTION)*) >
<!ELEMENT APPMAT        - -   (APPENDIX)* >
<!ELEMENT AUTHGRP       - -   (AUTHOR, AUTHOR) >
<!ELEMENT AUTHOR        - -   (FNAME, SURNAME | SURNAME, FNAME ) >
<!ELEMENT BACK          - -   (BIBLIST) >
<!ELEMENT BIBLIST       - -   (HEAD, (CITATION)*) >
<!ELEMENT BODY          - -   (CHAPTER)* >
<!ELEMENT BOOK          - -   (FRONT, BODY, APPMAT, BACK) >
<!ELEMENT CELL          - -   (P) >
<!ELEMENT CHAPTER       - -   (NO, TITLE, ((P)*, (EXAMPLE | FIGGRP)? )?, (SECTION)*) >
<!ELEMENT CITATION      - -   (TITLE, (AUTHOR)*, (CORPAUTH)*, OTHINFO?, DATE ) >
<!ELEMENT CORPAUTH      - -   (ORGNAME, CITY?) >
<!ELEMENT DFORMULA      - -   ( #PCDATA | FIGGRP, FIGGRP?) >
<!ELEMENT DOCUMENT      - -   (BOOK) >
<!ELEMENT EXAMPLE       - -   (P | TITLE, ((P | FIGGRP)*, ANSWER?)? ) >
<!ELEMENT EXERC         - -   (NO?, ((P)*, (FIGGRP, P? | ANSWER)? )? ) >
<!ELEMENT EXGROUP       - -   (TITLE?, ((EXERC)*, FIGGRP? )? ) >
<!ELEMENT FIGGRP        - -   ((FIG)*, TITLE?) >
<!ELEMENT FOREWORD      - -   (TITLE, (P)*) >
<!ELEMENT FRONT         - -   (TITLEGRP, FIGGRP, AUTHGRP, FIGGRP, FOREWORD,
                               PREFACE) >
<!ELEMENT ITEM          - -   (P)* >
<!ELEMENT LIST          - -   (HEAD, ITEM, (HEAD, ITEM)* | (ITEM)*) >
<!ELEMENT P             - -   ( #PCDATA | (DFORMULA | LIST | EMPH | FORMULA |
                               SUBSCR | SUPERSCR | FIGREF)*
                               (FIGGRP, FIGREF? | SECREF | SYMBOL)?
                               | FIGGRP | FORMREF | SECREF |
                               (SYMBOL, SYMBOL?)  | TABLE ) >
<!ELEMENT PREFACE       - -   (TITLE, (P)*) >
<!ELEMENT ROW           - -   (TSTUB?, CELL, CELL) >
<!ELEMENT SECTION       - -   (NO, TITLE, ((P | EXAMPLE | FIGGRP | LIST)*
                               (EXGROUP, (SUBSECT1)* | DFORMULA | (SUBSECT1)*)?
                               | (SUBSECT1)* )? ) >
<!ELEMENT SUBSECT1      - -   (NO, TITLE, ((P | FIGGRP | EXAMPLE)*, EXGROUP?)? ) >
<!ELEMENT TABLE         - -   (TBODY) >
<!ELEMENT TBODY         - -   (ROW)* >
<!ELEMENT TITLE         - -   ( #PCDATA | (SUBSCR)*) >
<!ELEMENT TITLEGRP      - -   (TITLE) >
<!ELEMENT TSTUB         - -   (P) >
```

Figure 1: A DTD for one textbook

2470	EN → H S
1325	EN → H
1056	EN → H S EX
995	EN → H TF S
549	EN → H I TF S
352	EN → H I CG S
258	EN → H I TF S EX
195	EN → H TF
138	EN → H I CG TF S
117	EN → H TF EX
97	EN → H I CG TF S EX
92	EN → H EX S
84	EN → H TF R
54	EN → H I PaF S EX
51	EN → H I CG S EX TF EX
46	EN → H I CG BW EX
44	EN → H I PrF
42	EN → H I EX S
34	EN → H I PaF S
34	EN → H I PI TF S
30	EN → H I TF TF S
29	EN → H I S EX S
28	EN → H I CG S EX TF S EX
24	EN → H S EX S
22	EN → H I PI R
21	EN → H R EX
21	EN → H S EX TF EX
20	EN → H EX TF S
1787	EN → H EX
1122	EN → H I S
1031	EN → H I S EX
574	EN → H I CG S EX
387	EN → H I EX
329	EN → H R
232	EN → H TF S EX
171	EN → H I R
125	EN → H I
100	EN → H PrF
94	EN → H I PI S
85	EN → H I CG R
66	EN → H I S EX TF EX
53	EN → H I TF R
47	EN → H I CG PrF
45	EN → H I S EX TF S EX
44	EN → H PI S
39	EN → H TF EX S
34	EN → H I CG PaF S EX
31	EN → H I S TF S
29	EN → H I II TF S
29	EN → H I BW EX
24	EN → H I CG EX
22	EN → H I R EX
22	EN → H TF TF S
21	EN → H S TF S EX
20	EN → H I CG R EX

Figure 2: Sample dictionary structures

thus determines how the DTD generation can proceed and which kind of quality can be expected from the result.

We also presented a generation tool we have designed and implemented which offers features for several application areas. The tool produces unambiguous content models that — after the basic generalization phase — may be further processed with both interactive and batch operations. These operations include partial trivialization of a model group, isolation of model groups, discovery of inclusions, and separation of common and rare cases.

References

[1] Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Technical Report ISO/IEC 8879, International Organization for Standardization ISO/IEC, Geneva/New York, 1986.

[2] Jinhua Chen. Grammar generation and query processing for text databases. Research proposal, University of Waterloo, January 1991.

[3] Keith Shafer. Automatic DTD creation via the GB-Engine and Fred. Technical report, OCLC Online Computer Library Center, Inc., 6565 Frantz Road, Dublin, Ohio 43017-3395, 1995. Accessible at URL: http://www.oclc.org/fred/docs/papers/.

[4] Sunniva M. K. Solstrand. Automatisk generering av DTD fra SGML-kodet materiale. M.Sc.thesis, Institutt for informasjonsvitenskap, Universitetet i Bergen, September 1994.

[5] Peter Fankhauser and Yi Xu. Markitup! An incremental approach to document structure recognition. *Electronic Publishing – Origination, Dissemination and Design*, 6(4):447–456, 1994.

[6] *Suomen kielen perussanakirja. Ensimmäinen osa (A–K).* Valtion painatuskeskus, Helsinki, 1990.

[7] Eve Maler and Jeanne El Andaloussi. *Developing SGML DTDs — from text to model to markup.* Prentice Hall PTR, 1996.

[8] ISO – International Organization for Standardization. *Information and documentation – Electronic manuscript preparation and markup, ISO 12083*, 1994.

[9] Helena Ahonen, Heikki Mannila, and Erja Nikunen. Generating grammars for SGML tagged texts lacking DTD. In M. Murata and H. Gallaire, editors, *Proceedings of the Workshop on Principles of Document Processing '94. Also to appear in Computer and Mathematical Modelling.*, 1994.

[10] Helena Ahonen. Generating grammars for structured documents using grammatical inference methods. PhD Thesis, Department of Computer Science, University of Helsinki, 1996. In preparation.

[11] Anne Brüggemann-Klein. Unambiguity of extended regular expressions in SGML document grammars. In Th. Lengauer, editor, *Proceedings of ESA '93*, Lecture Notes in Computer Science 726, pages 73–84. Springer–Verlag, 1993.