XML Mixed Content Grammars

Pim van der Eijk and Dennis Janssen

Cap Gemini Nederland B.V.

Abstract

Extensible Markup Language documents are composed of sequences of character data and embedded markup, under control of a Document Type Definition. We will argue that embedded markup needs to be dealt with at a syntactic level to account for interdependencies between linguistic structure and document structure, as found in style guides and Controlled Languages. Furthermore, we discuss the effect of XML content manipulation by NLP tools on XML document integrity, and outline conditions on grammars under which an NLP system preserves XML well-formedness and validity.

1 Introduction

Practical Natural Language Processing (NLP) applications, such as Controlled Language (CL) authoring support systems[1] and Machine Translation (MT) systems, are expected to process word processor documents that include punctuation and formatting information. Word processor file formats are typically (native) binary formats or (interchange) formats based on a marked up ASCII text format. The Microsoft Rich Text Format (RTF) is an example of a common markup format, which several commercial NLP systems support. The following fragment shows an RTF encoding of a string in which the word *bold* is put in bold face.

a piece of \bbold\b0 text in an RTF paragraph \par

This raises the question how, and at what level of linguistic analysis, embedded markup is to be accounted for. A common engineering approach to processing formatted text is to account for markup at the levels of *text analysis* and/or *lexical analysis*. A text analyzer (often specific for a particular input file format) encodes markup using a kind of escape syntax. The lexical analyzer skips the escaped items, but anchors them to nearby linguistic items. After syntactic processing, the markup is put back into the generated output string at the location of the (translation of the input) text anchors.

Systems based on this approach include several commercial MT systems and earlier versions of Cap Gemini's Controlled Language authoring system (van der Eijk, de Koning and van der Steen 1996), marketed as *Cap Gemini CLarity* and referred to as CLarity from now on. In CLarity, an additional distinction is made between prefix markup (such as **\\b**) and postfix markup (such as **\\b0**), and between word-level markup and sentence/paragraph-level (**\\par**) markup. The prefix/postfix distinction is also used to account for Extensible Markup Language (XML) open and close tags (see section 2).

[1]See Adriaens, Havenith, Wojcik and Tersago (1996) and Mitamura and Nyberg (1998) for recent work on Controlled Language applications, and van der Eijk (1997) for a general introduction.

This approach is clearly a very useful starting point, as it prevents the user from having to reapply the original formatting to NLP system output. However, it is based on two assumptions which we will argue to be incorrect. First of all, it is assumed that the linguistic content of a document can be processed independently from the markup. This assumption is already questionable in the case of formatting markup (e.g. a text-to-speech engine might generate an audible emphasis for bold or italicized text), and clearly invalid for uses of SGML and XML markup for *content tagging*. For instance, in the following procedure step (drawn from an Interactive Electronic Technical Manual), the content tag *Button* provides information that *Next* is a proper noun rather than an adverb:

> <**Step**>Push <**Button**>Next</**Button**> for generator repair instructions.</**Step**>

The design philosophy underlying the Standard Generalized Markup Language (SGML) concept of *generic markup* is that *document content* is a combination of textual content and markup, where the markup provides essential structural information, inline metadata content, and links to external multimedia data. In section 3, we will elaborate on this issue and argue that a proper computational account of style guides and Controlled Language (CL) specifications requires manipulation of *structured content*, and that viewing these specifications as a kind of sublanguage specifications is an inherently limited perspective.

A second problem with the treatment of markup at the text/lexical analysis levels is the issue of *document integrity*. The SGML concept of *validation* implies that the distribution of markup in a document is controlled using Document Type Definitions (see section 2.2). NLP systems that manipulate (e.g. reorder) their input, including markup, might thus produce invalid document structure. As an example, assume the following partially content-tagged input sentence:

> <**Para**> The conference is held on <**date**><**day**>11</**day**> december 98</**date**></**Para**>

The existing lexical analyzer of CLarity would anchor *<date><day>* as markup prefix of *11*, *</day>* as its postfix and *</date>* as postfix for *98*. A (hypothetical) date normalizer would then produce the following alternative, incorrectly moving the word *December* outside the content of the *date* element.

> <**Para**>The conference is held on December <**date**><**day**>11,</**day**>1998< /**date**></**Para**>

This shows that the content tag *date* needs to be treated at a (phrasal) syntactic level rather than a lexical level. Also note that, if *December* is tagged as *<month>December< /month>* in the input sentence, the *month* element would be moved outside the *date* element. Depending on whether the DTD allows *month* elements as content of *Para* elements, the application of the date normalizer might then invalidate the document.

In this paper, we will outline an approach to processing structured documents that accounts for document structure at the level of syntactic analysis. As this approach requires some understanding of document technology concepts, section 2 will first provide a brief introduction to some concepts and jargon from structured document processing (SGML/XML).

The issue of accounting for *structured content* will be taken up in section 3, which will provide two case studies that require integrated content processing. In section 4, we will discuss a recent generalization of CLarity to a system for *controlled content* creation, and will show how this allows us to more accurately and completely implement Controlled Language specifications and to provide better support to authors of Controlled Language content.

The issue of accounting for *document integrity* will be discussed in section 5, where we will outline a number of conditions under which controlled content manipulation systems will preserve the two types of document integrity distinguished in the XML standard. Section 6 will discuss related work. Section 7 will summarize the results of this paper and will outline some potential extensions.

2 Structured Documents

2.1 Extensible Markup Language

Extensible Markup Language (XML) is a representation language for structured documents defined by the World Wide Web Consortium (W3C) for Internet applications (Bray and Paoli 1998). It is derived from Standard Generalized Markup Language (SGML), an International Standard (ISO 8879). XML retains the core SGML concepts of *generic markup* and *validation* but eliminates the engineering complexity of SGML and a number of rarely used or less well-designed properties of SGML.

XML offers an object-oriented approach to modelling and processing document content. Documents are viewed as tree-structured assemblies of individual *logical* information components, called *elements*, and are represented physically using a markup syntax. Elements can either be empty or have *content*. Non-empty elements are marked up using a *start tag* and an *end tag*. For instance, the *Hello, world* string is enclosed by open and close tags of the *greetings* element. Empty elements, as the `<br/>` example, are marked up using a single tag that has a special syntax:

> **<greetings>**Hello, world< **/greetings>**
> there's a line **<br/>** break in this line

Elements that have content can be subdivided in three subclasses. Many elements, typically including the top-level document element, are containers for other elements. For instance, the element *html* in the HTML DTD consists of two elements, viz. a *head* and a *body*. This case is called *element content*. A hypothetical element *CardNumber*, which might occur in an E-Commerce application to tag a credit card number, would not contain subelements but would only contain a string of digits. This is called *data content*.

<html><head><!– head content ...–>**</head><body>**<!– body content ...–>**</body></html>**

<CardNumber>5437 4027 5726 5306**</CardNumber>**

Many elements contain a mixture of running text with embedded subelements. An example of this is the IETM generator repair instruction we discussed in section 1. This fragment can be thought of as representing a (sub)tree with a mother node labelled *Para* that has three daughter nodes, viz. an embedded subelement labelled *Button* with data content *Next*, which separates two untagged nodes containing data content, the strings *Push* and *for generator repair instructions*, respectively. Such combinations of data content and subelement content are called *mixed content*. A data content node in mixed content is sometimes referred to as a *pseudo element*.

2.2 Document Type Definitions

In XML, a class of documents is modelled using the Document Type Definition (DTD) modelling language. DTDs are used by content entry systems to support guided authoring, and can restrict the user to insert only those subelements or data content that are allowed at a particular location in the document. DTDs are also used as interface specification in information or data exchange applications.

The distinction between data content elements, mixed content elements and element content elements is accounted for in the DTD by *element definitions*. The examples of elements that illustrate the three types of content given in section 2.1 can be defined as follows.

```
<!ELEMENT br EMPTY>
<!ELEMENT CardNumber (#PCDATA)>
<!ELEMENT html (head, body)>
<!ELEMENT Para (#PCDATA|Button)*>
<!ELEMENT list (head?, ((label, item)+ | item+))>
```

The last of these definitions is an element that illustrates a more complex type of element content definition, the TEI Lite (Burnard and Sperberg-McQueen 1995) list element.

The description of the allowed content of an element is called the *content model*. All XML element definitions follow the pattern *<!ELEMENT ElementName ContentModel>*. Empty elements have the fixed content *EMPTY*. The content model of all elements that have data content is *#PCDATA*.

Content models of elements that have element content are constructed from element names combined using unary and binary operators. These operators express (with *X, Y* valid content model sub-expressions) linear order (*X* , *Y* means X followed by Y), alternatives (*X* | *Y* means X or Y), optionality (*X* ? means optionally X), and iteration (the * and + operators with the usual interpretation). Parentheses can be used in content models for grouping. The *list* element definition thus states that *list* elements may but need not contain an initial *head* element, and then consist of either a sequence of pairs of a *label* element and an *item* element, or a sequence of one or more *item* elements.

In XML, the definition of elements that have mixed content is fixed to be the application of the * operator to a single group of *#PCDATA* and element names combined using the | operator. *#PCDATA* is required to be the first item in such a |-separated list.

Any SGML document instance can be converted to an XML data stream without loss of information (other than encoding information). As a result of this, focussing on XML rather than SGML means no loss of generality. However, XML mixed content rules cannot further constrain the relative ordering of subelements and data content in mixed content. SGML DTDs lack this restriction, instead *#PCDATA* can be used alongside element names in complex content models. SGML DTDs are thus more expressive than XML DTDs. The motivation for the special restriction on mixed content models is beyond the scope of this paper and unrelated to the topic of integrating XML processing and Natural Language Processing. However, we will show that the XML restriction allows us significant simplifications in section 5.

2.3 Document integrity

The XML standard defines two types of integrity properties that can hold true of a particular XML data stream, viz. *validity* and *well-formedness*.

The XML Document Type Definition mechanism can be used to determine whether a particular XML data stream is *valid*, as the content of each element can be verified against the content model defined in the DTD. This is essentially parallel to context-free grammar parsing.

A less strict constraint than validity is *well-formedness*, which requires a tagged data stream to contain a proper nesting structure of open tags and close tags (and involves a number of additional constraints not relevant to this paper). Any document that is valid is well-formed, but not vice versa. The concept of well-formedness was introduced for applications that do not need to validate the XML data stream.

Section 5 discusses a number of conditions on grammars that make sure that the CLarity run-time engine does not produce alternatives for valid XML input that are not well-formed or turn the XML document invalid.

3 Style guides and Controlled Language

3.1 Authoring support systems

Our interest in processing structured documents is the commercial application of language technology to support authors to create high-value documents such as technical documentation for complex industrial or military systems, requirements specifications, or Service Level Agreements (SLA). These are documents that have to satisfy high quality requirements such as comprehensibility, consistency, reduced ambiguity, improved translatability, and improved machine-processability.

Style manuals and Controlled Language specifications provide (usually rather informal) information how these requirements can be met. Currently, a number of research and commercial systems offer interactive or batch functionality to support authors or editors of CL text. These systems are similar to grammar checkers, dedicated to and customized for a specific application domain or customer, and offer critiques and possibly alternatives for sentences that fail to meet specific criteria.

It can be argued that viewing style guides and Controlled Languages as a kind of prescriptive sublanguages (van der Eijk et al. 1996) does not do full justice to these documents. In fact, as the title of MIL-PRF-87268 (*Interactive Electronic Technical Manuals - General Content, Style, Format and User-Interaction Requirements* 1995) shows, these documents typically address not only syntactic and lexical issues, but also provide guidelines for high-level, rhetorical structure (how to convey certain types of information), address low-level issues like formatting and punctuation, and discuss the proper use of document structure elements, including multimedia elements.

To illustrate this, we will provide evidence from two case studies.

3.2 Microsoft Manual of Style for Technical Publications

The *Microsoft Manual of Style for Technical Publications* (Microsoft 1995) is a style guide for software documentation. Four rules specified in this document are presented here:

> Do not add punctuation at the end of the heading.
>
> Avoid beginning a heading with an article (*a, an, the*).
>
> In end-user task-oriented documentation, generally use a present participle (*ing* form) rather than an infinitive to begin headings, except for procedure headings, which should begin with an infinitive phrase.
>
> As a general rule, tell the user where the action should take place before you describe the action to take. This prevents the user from doing the right thing in the wrong place. [..] For example, the following phrase is typical: *On the* **View** *menu, click* **Zoom**.

The first three of these rules illustrate a number of interdependencies of linguistic structure (use of articles, verb form, capitalization, punctuation) and document structure. A support system needs to determine document context to provide accurate author feedback in these cases. The fourth rule is an example of a rhetorical rule.

3.3 Simplified English

The Simplified English (SE) standard is a controlled language for the aerospace industry (AECMA 1995). It contains a controlled vocabulary and a set of writing rules. Companies in this industry are interested in computer support to help them check compliance to this standard. Some examples are given here:

RULE 2.3 When appropriate, use an article (the, a, an) or a demonstrative article (this, these) before a noun.

RULE 4.3 Use a tabular layout (vertical layout) for complex texts.

WRITE:
the controls of the main panel, from top to bottom, are:

- An OFF/ON main switch
- A START push button
- A STOP/O.S. TEST push button

NOT:
From top to bottom, the controls on the main panel consist of an OFF/ON main switch, a START push button, and a STOP/O.S. TEST push button.

To implement SE rule 2.3, it is possible to write a grammar correction rule that inserts an article in Noun Phrases that lack an article. This rule needs to account for the fact that *appropriate*ness of a context for article insertion not only depends on linguistic criteria (such as the count/mass noun distinction, generic/specific interpretation of plurals etc.), but also on document contexts. For instance, articles are generally inappropriate in *headings* and table *cells*.[2]

In the discussion of rule 4.3, an example is given where *text content* (a complex NP enumeration) is replaced by insertion of a *list* element. The functionality described in section 4 shows how such a transformation can be implemented in an authoring support tool like CLarity. This functionality thus generalizes CLarity from a Controlled Language tool into a controlled *content* tool. This rule is discussed in more detail in section 4 and 5.

4 Mixed Content Grammars

The discussion in section 3 shows the need to process markup at a syntactic level. A controlled content system should be able to *manipulate* (recognize, delete and insert) markup as it manipulates textual content to produce alternatives for input that violates one or several Controlled Language rules. This means there is a need for a unified data model for document structure and linguistic structure, and a unified language for modelling the set of valid document structures and linguistic constructions.

The linguistic coverage of CLarity is defined using an extended context-free grammar formalism augmented with an attribute unification constraint language. The formalism also supports annotations on grammar rules that are interpreted as

[2]The absence of articles in specific document contexts also showed up in a prototype Dutch-to-French Machine Translation system developed at Cap Gemini in 1995. Whereas article omission in French is less common than in Dutch, articles are generally omitted in table cells and headings in French.

instructions to transform input expressions to output expressions. These annotations to grammar rules express reordering, insertion and deletion operations on parse trees.

Well-formed XML documents can also be represented as labelled trees with attribute specifications. The XML DTD language for document structure can be viewed as a variant of the context-free grammar formalism.

Therefore, there is no essential difference between the XML and CLarity representations and modelling languages. In practice, the level of (linguistic) analysis performed by NLP systems is (much) more fine-grained than the document structure hierarchy. Theoretically, CLarity could be used as an XML validator by providing it with a grammar that mirrors the DTD hierarchy, with an additional rule for *#PCDATA* (which accepts any lexical item). The DTD can then be considered to express a (database) *view* on a more complex structure definition.

In practice, CLarity and similar Controlled Language authoring tools are intended to be called as plug-ins to existing authoring systems, such as SGML/XML editors. Therefore, the document fragments that need to be passed to the CLarity system are the substructures contained within mixed content and data content elements. As implemented, XML data streams (rather than tree data structures of the host XML editor) are passed between engine and plug-in. These streams thus consist of mixed markup and text.

The set of fragments that are to be passed to the linguistic engine can either be application-dependent, or derived from the content model of the element as specified in the DTD. When a mixed content element or data content element is contained within another mixed content element, the *larger* fragment needs to be processed if the content of the embedded element is constrained linguistically by the content of the larger fragment, or vice versa. In that case, either the document element has mixed or data content and its full content is passed to CLarity for processing, or the document is divided in a number of fragments, where each fragment corresponds to the content of a mixed or data content element, any ancestors of which up to the document element are element content elements.

CLarity has been extended to support syntactic processing of XML streams. The specific approach adopted aimed at achieving maximum flexibility for linguistic developers with minimal changes to existing software. The *software* changes are made in the text analysis and lexical analysis modules. The *linguistic* changes affect the Controlled English grammar and lexicon and are specific to a particular XML application. Both grammar and lexicon need to be augmented with, for each non-empty XML element *E*, two additional terminal categories *XML_E_start* and *XML_E_end*, and lexical entries <*E*> and </*E*> that have these categories, and analogously for empty elements. The adapted lexical analyzer maps XML markup to these lexical entries. Attributes on XML elements can be mapped to attributes on the corresponding grammar categories.

An example rule that transforms

<Heading>The generator.**</Heading>**

to

<Heading>Generator**</Heading>**

consists of a rule that recognizes an *XML_Heading_start*, an NP, an optional period and an *XML_Heading_end*. The rule has an annotation that deletes the period, if present. Using an attribute, information is passed into the NP to signal that any article, if present, is to be deleted. This information is passed from NP, via intermediate grammar rules for determiner phrases, down to the article rule, which deletes (or inserts) the article depending on the particular value for the attribute that passes the correction information. This rule is an example of a *structure-sensitive* rule. From the XML point of view, there is only a minor *#PCDATA* change, the markup tree structure is essentially preserved without changes.

A more interesting rule is concerned with AECMA rule 4.3. This rule can be implemented by modifying the rule that recognizes enumerations of NPs to insert *list* and *item* XML open and close tags. The following instruction (taken directly from the XLA 115 Special Purpose Generator manual shipped with the HyMID IETM system (Anderson 1996)) is an example of a sentence that would match this rule.

To clean carburetor ports, clean$_{a/v}$ channels and discharge$_{n/v}$ ports

The part of speech subscripts indicate that this sentence is at least three times ambiguous. This may potentially cause misinterpretations. Application of AECMA rules 2.3 and 4.3 will yield a number of unambiguous alternatives, one of which is the following:[3]

To clean **the** carburetor ports:**<List><Item>**Clean **the** channels**</Item><Item>**Discharge **the** ports**</Item></List>**

The user only needs to select the intended interpretation and paste it into the document.

Note that the ability to process XML-encoded data in CLarity is not dependent on an integration with an SGML/XML editor. In fact, some of this functionality is currently used in a Controlled English editor embedded in Microsoft Word (van der Eijk 1997), where the (limited) structural information that is available in Microsoft Word (heading styles, lists, cells) is encoded in XML for communication with the underlying CLarity engine.

4.1 Other XML auto-tagging applications

Other applications of inserted markup include XML tags to mark content for further processing. For instance, if on the basis of lexical and grammatical information it can be determined that *auxiliary power unit* is a term to be indexed in:

Switch on the auxiliary power unit

[3]Note that the specific set of tags used to encode lists is DTD-specific.

Then tags could be inserted around the phrase so that the index generator can record this occurrence of the term. This is an example of XML *auto-tagging*. This type of auto-tagging is based on a fairly sophisticated natural language analysis, and thus is potentially more precise (though obviously less robust) than the conventional pattern matching techniques used in the SGML/XML community.

Switch on the <**Term**>Auxiliary Power Unit</**Term**>

Other uses of inserted tags include the use of tags to record user-supplied word sense disambiguation information, as done in the Controlled Language project at Caterpillar (Kamprath, Adolphson, Mitamura and Nyberg 1998).

5 Document Integrity

5.1 Introduction

As discussed in section 2.3, XML distinguishes two types of integrity constraints: well-formedness and validity. The incorporation of XML processing within general Controlled Language processing described in section 4 offers the grammar writer the possibility to specify arbitrary reordering, deletion and insertion of XML markup.

Nothing inherent in the formalism prevents a linguist from specifying a grammar that accepts XML input or generates XML output that violates one or both of these constraints. As the system is designed to be called from a validating SGML/XML editor (which ensures that the user cannot enter invalid content), we are especially interested in preventing rules that accept valid XML input but produce invalid, or even non-wellformed XML output. Note that validity is a special case of well-formedness.

In this section, we will focus on the extended context-free core, and the mechanism to produce corrected output. The proof sketched in this section carries over to the actual, significantly more complex formalism used to specify the CLarity Controlled English grammar.

5.2 A grammar formalism for correction

The CLarity formalism is a context-free grammar formalism extended for optional, alternative and repeated right-hand side symbols, attribute constraint evaluation, and morpho-syntactic and word order correction. This grammar specifies a set of pairs of *input* language expressions and associated *corrected* output expressions. The association is expressed at the level of context-free grammar rules using additional annotations. These annotations can perform the following operations:

Reorder a subnode

Delete a subnode

Insert a lexical item.

Note that reordering and deletion operate at the order of complete subnodes, and that an inserted lexical item is inserted at the same level as the other subnodes of a rule. A subnode is reordered including all of its content. An inserted lexical item can be a textual item (such as an inserted article), or markup (such as the close tag of *item*, the lexical item **</item>**).[4]

5.3 Well-formedness

The requirement of well-formedness, for the purpose of this discussion, will be limited to the following requirements:

1. Open and close tags of non-empty elements must be in that order (*Properly Sequenced*)
2. Open and close tags of non-empty elements are allowed within other pairs of open and close tags but with no crossing dependencies. (*Properly Nested*)

We will show that when a grammar satisfies the following conditions, CLarity will generate well-formed (possibly corrected) XML output, if the XML input was valid to begin with:

A Pairs of Open and Close tags must be used in the same rule

B Pairs of Open and Close tags may not be reordered within a rule in such a way that the close tag precedes the open tag.

C Pairs of Open and Close tags can be inserted or deleted in pairs.

D Pairs of Open and Close tags are properly nested w.r.t. other pairs of open and close tags in that rule.

These conditions can be interpreted as either guidelines for grammar writers or verified using a static analysis of the grammar.

Theorem Let I be a Language (a set of XML input strings) that is XML well-formed. A Clarity grammar G that complies with conditions **A**, **B**, **C**, and **D** and accepts Language I produces a Language O (Corrected Output Sentences) that is XML well-formed.

The proof is based on induction on the number of rule applications in a parse tree. The *base case* is concerned with parse trees that consist of a single rule application. These parse trees only have leaf nodes. These nodes can be any (mixture) of the following:

[4] In the extended formalism, lexical choice of inserted elements can be further constrained using attribute unification. At the level of terminal nodes, there are other types of correction (e.g. agreement correction, lexical preference relations) that are irrelevant to the discussion. These corrections do not change the category of the terminal node, and thus preserve well-formedness and validity.

Non-markup terminal nodes These terminal nodes correspond to *#PCDATA* data content. Data content does not interfere with XML well-formedness.

Empty elements As these elements by definition are represented using a special tag, proper sequencing and nesting of open and close tags is not affected. There are therefore no restrictions on reordering of empty element tags.

Content elements These elements come in pairs of an Open tag and a Close tag, which will be properly nested and sequenced, because:

by condition **A**, for an open (close) tag there is a corresponding close (open) tag,
by condition **B**, these are Properly Sequenced,
by condition **C**, if one of the tags is deleted (inserted) both will be
by condition **D**, if there are other pairs of open-close-tags these are Properly Nested.

The *induction part* of the proof is based on the assumption that the theorem is proven for parse trees containing less then n rule-applications. We then have to prove it for parse trees with $m \geq n$ rule-applications.

The general scheme for a syntactic rule r that produces a parse tree with top node P out of parse nodes $P_1 \ldots P_z$, with $1 \leq i \leq j \leq z$ has the following form:

$$P = P_1 \ldots P_i \ldots P_j \ldots P_z$$

Each of the subtrees P_i can be annotated with annotations for deletion, reordering to a position j different from i, or insertion at a location i. Assume all P_i contain less then n rule-applications. We need to distinguish the following cases:

Node reordering We need to distinguish two cases:

1. *Complex subtrees P_i* can be reordered without affecting well-formedness. This operation does not affect pairs of open/close tags within the subtree, which thus remain Properly Sequenced and Properly Nested.

2. *Leaf nodes P_i* are either non-markup leaf nodes (XML *#PCDATA*), a node representing an empty element tag, or an open (resp. close) tag for a content element. The first two of these cases have no impact on well-formedness. As the grammar complies with conditions **A**, **B** and **C**, for each open tag P_i (or close tag P_j) there is a corresponding close tag P_j (respectively, open tag P_i) such that $i < j$, and such that P_i and P_j are also Properly Nested in the output XML structure.

Node insertion Only leaf nodes can be inserted. Insertion of non-markup leaf nodes or an empty element tag leaf node does not affect XML well-formedness. For an open tag leaf node P_i (respectively, a close tag leaf node P_i) that is inserted, by condition **C** there is a close tag leaf node P_j (respectively, an open tag leaf node P_i), where $i < j$.

Node deletion Here we need to distinguish deletion of leaf nodes from deletion of complex subtrees. The proof that deletion of leaf nodes preserves well-formedness is analogous to the case of node insertion. By condition **A**, there will remain a corresponding close tag P_j for each open tag P_i (and vice versa), where condition **B** ensures that $j > i$ (Properly Sequenced), and condition **D** ensures these are Properly Nested w.r.t. other pairs of open and close tags in the rule.

This proof for context-free grammars carries over to extended context-free grammars containing operators for optionality, iteration and or-expressions, by requiring that conditions **A** to **D** apply to each rule sub-expression in the scope of such an operator.

5.4 Validity

As discussed in section 4, the CLarity engine is invoked by an authoring plug-in to process the content of an XML element that has either mixed or data content.

In section 5.3, we have provided a number of conditions on grammars that make sure the generated output will always be *well-formed*. Our longer term objective is to establish a set of additional conditions on grammars that ensures that all generated XML output satisfies XML *validity* as specified in the XML Document Type Definition. Analogously to the discussion of well-formedness, this would obviate the need to dynamically validate generated XML output fragments. For the moment, we sketch an interim result which covers the special case where all elements contained within the fragments passed to CLarity have either mixed or data content, but not element content. This property may hold true of a specific DTD. The corrected XML data stream will then still need to be validated using an external XML validator if some element contained in the input data stream is declared in the DTD to have element content.

Analogously to the discussion of well-formedness, we will distinguish the three operations of node reordering, node deletion and node insertion, and only consider manipulation that follows conditions **A**,**B**, **C** and **D** of section 5.3 and thus maintains well-formedness. Note that XML elements that have mixed content or data content can only specify *which* subelements (if any) can occur as content of elements in addition to *#PCDATA* , but cannot specify additional order or occurrence constraints.

Node reordering might rearrange some input string <Z> <*X*/> ... <*Y*/> </Z> into <Z> <*Y*/> ... <*X*/> </Z> . If Z is declared as having mixed content, the XML constraint on mixed content models (discussed in section 2.2) ensures the result will not just be well-formed, but also valid. Similarly, as XML DTDs cannot enforce presence of certain subelements, *node deletion* cannot have any effect on validity. The only special case to be considered is *insertion* of markup leaf nodes. The specific element that is inserted may or may not be allowed according to the DTD.

One approach to checking a grammar against a DTD uses the following procedure. First, we determine for each element E the set of allowed subelements CM.

This set is $\emptyset$ if the element has data content and is $\{ S_1 \ldots S_N \}$ if the element has a mixed content model *<!ELEMENT E (#PCDATA* | S_1 | ... | S_N*)⋆ >* .

For each grammar rule where an XML subelement is inserted, the insertion context can be determined. If the nearest containing element *C* is specified directly in the rule, the validity can be determined locally, at the level of the individual rule.

$$P = P_1 \ldots < C > \ldots < \mathbf{E}/ > \ldots < /C > \ldots P_z$$

If the grammar rule does not itself specify an enclosing element, a static analysis of the grammar must be performed to determine, not a container for <E/>, but rather a containing element for *P* (possibly even for an ancestor of *P*). Since there may may be several such containing elements, to check whether element *E* is allowed in all such contexts, it should be a member of the intersection of all CM sets of these elements.

As an example, the rule that implements SE rule 4.3 is a modified rule for enumerations of NPs. This rule does not explicitly mention an enclosing XML element, but is used in sentences that occur inside a *Para* element or within a *Heading*. Assume the DTD allows sublists inside paragraphs but not in headings. In that case, the *List* element is not included in the intersection of CM sets, meaning the rule as stated overgenerates.[5]

Note that this approach only applies to inserted subelements. It does not help the grammar writer detect cases of incorrect reference to tags that are used contextually. The approach also cannot easily be extended to element content elements or SGML mixed content, where the relation between document structure and linguistic structure is more complex. That type of functionality would require a more explicitly algebraic approach to modelling the relations between linguistic structure and document structure. One potential approach would be based on the HyTime concept of *architectures* (Goldfarb, Newcomb, Kimber and Newcomb 1997).

6 Related work

Controlled Languages are typically used in technical publication departments of (large) companies, which are also major users of SGML-based authoring and publication systems. Several companies currently offering commercial Controlled Language products and/or services have developed interfaces with SGML editors. Approaches similar to the one discussed in this paper have therefore been developed independently in a number of projects. Some of these projects have been discussed in the literature, including SECC (Adriaens 1996), the KANT application at Caterpillar (Kamprath et al. 1998) and Siemens-Dokumentationsdeutch (SDD), a Controlled German project (Schachtl 1996). The SECC and SDD papers discuss processing *#PCDATA* sensitive to the encompassing data content element. In section 4.1, we discussed the use in the Caterpillar KANT application of markup to encode disambiguation information.

[5]A grammar writer can then create different rules for the two contexts. In practice, the grammar writer would distinguish the two cases using attributes that encode contextual restrictions.

Research at Aerospatiale points at the need to be able to modularize Controlled Language design for specific document classes (Lalaude, Lux and Regnier-Prost 1998). A system like the one presented in this paper would seem to be a useful implementation framework for these ideas.

To our knowledge, this article is the first published discussion of the importance of document integrity in relation to NLP, and of conditions on grammars and DTDs that ensure XML document integrity is maintained during linguistic processing.

7 Conclusion

In this paper we have argued that style checkers and Controlled Language tools need to go beyond syntactic checking and generation of critiques and alternatives and need to evolve into *controlled content* processing. XML is a generic content representation language which is rapidly becoming popular for both document and data representation. In this paper, we discussed a simple and effective approach of extending an existing Controlled Language tool, CLarity, to process XML encoded documents. This approach involves mapping XML DTDs to a constraint-based grammar formalism. With minimal changes to the existing Controlled English grammar, the resulting system is now able to support a number of AECMA Simplified English rules that hitherto could not be implemented correctly. The proposal can be generalized to support arbitrary DTDs.

A drawback of the approach is that modularity of grammar and DTD is lost, and that grammar developers need to be both competent in computational linguistics and in document technology. In practice, the modularity issue can be worked around using software engineering techniques like conditional compilation, but maintenance may become an issue if several complex, different DTDs are to be supported.

Since the level of linguistic analysis performed by systems like CLarity is typically much more fine-grained than XML document modelling, our approach has been based on embedding XML document structure processing within the syntactic analysis of an existing Controlled English grammar (see section 4). This raises the issue of making sure that the manipulation of content (text and markup) by the CL system maintains XML document integrity constraints. We have shown in section 5 that this requirement can be expressed as a number of conditions on grammars. These conditions can be interpreted both as guidelines for grammar writers and implemented as a verification procedure that operates as a static check on a grammar and DTD. Note that these conditions completely obviate the need for dynamic validation of the output of the CL system at run-time.

References

Adriaens, G.(1996), SECC: using text structure information to improve checker quality and coverage, *Proceedings of the first international workshop on controlled language applications (CLAW96).*

Adriaens, G., Havenith, R., Wojcik, R. and Tersago, B. (eds)(1996), *Proceed-*

ings of the first international workshop on controlled language applications (CLAW96), Katholieke Universiteit Leuven.

AECMA(1995), *A Guide for the preparation of aircraft maintenance documentation in the international aerospace maintenance language*, European Association of Aerospace Industries, Brussels. PSC-85-16598.

Anderson, M. (ed.)(1996), *The Metafile for Interactive Documents. Application Guide and Draft Performance Specification for the Encoding of Interactive Documents*, Naval Surface Warfare Center, Carderock Division, Maryland.

Bray, T. and Paoli, J. (eds)(1998), *Extensible Markup Language*, World Wide Web Consortium. W3C Recommendation.

Burnard, L. and Sperberg-McQueen, C.(1995), TEI lite: An introduction to text encoding for interchange, http://www-tei.uic.edu/orgs/tei/intros/teiu5.html.

Goldfarb, C., Newcomb, S., Kimber, E. and Newcomb, P. (eds)(1997), *ISO/IEC 10744:1997 Hypermedia and Time-based Structuring Language*, International Standardization Organization.

Interactive Electronic Technical Manuals - General Content, Style, Format and User-Interaction Requirements(1995), Department of Defence. Performance Specification MIL-PRF-87268A.

Kamprath, C., Adolphson, E., Mitamura, T. and Nyberg, E.(1998), Controlled language for multilingual document production: Experience with Caterpillar Technical English, *Proceedings of the second international workshop on controlled language applications (CLAW98)*.

Lalaude, M., Lux, V. and Regnier-Prost, S.(1998), Modular controlled language design, *Proceedings of the second international workshop on controlled language applications (CLAW98)*.

Microsoft(1994), *Rich Text Format (RTF) Specification*, Microsoft.

Microsoft(1995), *Microsoft Manual of Style for Technical Publications*, Microsoft Press.

Mitamura, T. and Nyberg, E. (eds)(1998), *Proceedings of the second international workshop on controlled language applications (CLAW98)*, Carnegie Mellon University.

Schachtl, S.(1996), Requirements for Controlled German in industrial applications, *Proceedings of the first international workshop on controlled language applications (CLAW96)*.

van der Eijk, P.(1997), Controlled languages in technical documentation, *Computational linguistics in the Netherlands 1997*.

van der Eijk, P., de Koning, M. and van der Steen, G.(1996), Controlled language correction and translation, *Proceedings of the first international workshop on controlled language applications (CLAW96)*.