A grammar view for editing structured documents

Mark Sifer
School of Computer Systems Engineering
University of New South Wales
Sydney, New South Wales, Australia
msifer@cse.unsw.edu.au

Yardena Peres and Yoelle Maarek
Knowledge Management Group
IBM Research Lab in Haifa
Matam, Haifa, 31905, Israel
{yardena, yoelle}@il.ibm.com

Abstract

We propose simplifying the editing of structured documents that conform to Backus-Naur Form (BNF) grammars, by exposing and operating on the grammar itself. We introduce an original grammar view that supports browsing and editing of structured documents and is coordinated with the document view. The grammar view presents document element types in context to better support document editing. Both shading and font size scaling are used to create a focus area in the grammar view making it more scalable for larger grammars. We present an implementation of a grammar view for XML documents, in our XML editor, *Xeena for Schema*.

Keywords: User interface, structured document, coordinated views, XML.

1 Introduction

Structured documents come in a number of forms, from SGML for large document collections to XML its recent Web derivative. These documents are typically text files whose content is delimited and structured with markup tags. In the case of XML, most documents conform to a BNF grammar. Many XML grammars have been defined including for mathematics, graphics, Web maps etc.

We are concerned here with the problem of authoring such structured documents that have a predefined grammar. A typical approach for many XML editors, (e.g. XML Spy) is to represent document content as a tree of elements. Editing operations are then supported by a context sensitive palette of element types, which changes according to the target position of the considered type in the document. When the element types are numerous or the user is not familiar with the documents, choosing types and placing them in a document becomes pretty difficult. We propose to tackle this problem by visually exposing the grammar to which the structured document must conform and directly operate on the grammar to browse and edit documents. In other terms we use a grammar view rather than a palette. Our main goal is to support an intuitive user-interface paradigm for editing operations even when not familiar with the particular grammar. This solution has been embodied in a Java based XML editor that we describe in the following.

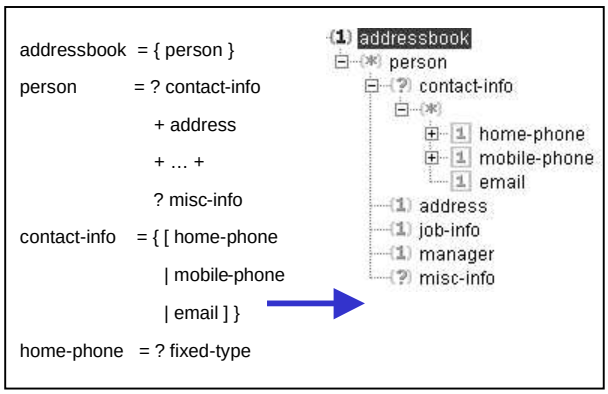

Figure 1: Text and Graphic Views of a Grammar

A structured document, that conforms to a BNF-based grammar, can always be represented by a tree. We propose here to take advantage of this property and expose the grammar to users in a tree view. Figure 1 shows such a tree representation of a BNF grammar, that we call "grammar view" hereafter. This view is used for supporting various operations on structured documents that conform to the grammar.

2 Browsing with a Grammar View

The structured document and associated grammar views are shown side by side (on the right and left hand side respectively in Figure 2). Operations in both views are coordinated and several focus techniques are used such as shading and font size scaling to create a focus area.

Each grammar type is preceded by a grammar icon, which consists of either a bracket or a box that encapsulates a symbol. The symbols can be "?" , "*" ,or the values "1" or "n". A surrounding bracket indicates that a given type and its siblings form a sequence, while a box indicates a choice. A "1" indicates an compulsory occurrence, a "?" indicates an optional occurrence, and "*" denotes that zero or more occurrences may appear. The grammar view differs from the document tree in two ways. In the grammar view, all possible child types appear whether they are sequences, choices or optional whereas only the instantiated one appears in the document view. Similarly a n- or *- labelled type in the grammar view (e.g., the "person" type) appears once while it may be repeated in the document view. When a user selects an element in a document view, the grammar tree dynamically changes and selectively expands or restricts some types in the grammar. In our example, if the cursor is positioned on "home-phone" in the

document view, the associated grammar will include its associated type, as well as the types of its siblings, as well

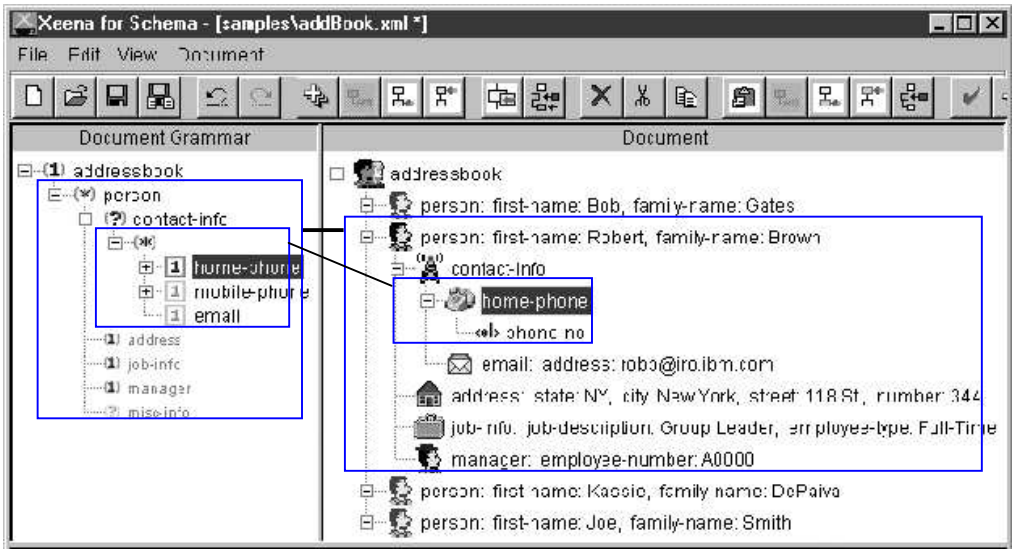

Figure 2: Text and Graphic Views of a Grammar

as the types of all of its ancestors and ancestors siblings up to the root so as to maintain context. In some cases, child types may also be included. When a type has no corresponding document node (e.g., the optional "misc-info") its grammar icon is grayed out and shaded

3 Editing with a Grammar View

All elements are instantiated through the grammar view. For example, to add a "misc-info" element for Robert Brown the user would select the misc-info type then click the add toolbar button (or just shift-click). The colouring of icons in this view sets the context for editing operations. Thus, the above misc-info element can be added only if grayed out in the grammar view, as the grammar allows for at most one occurrence of that type in the document. Less relevant types are grayed and shown in a smaller font according to their closeness to the type currently selected. We have annotated Figure 2 to explain the mapping between the grammar and the document views. When selecting the person element "Robert Brown" in the document view, the associated type is dynamically highlighted and becomes the focus of the grammar view. Conversely, the elements of the same type person in the document view are switched to "black and white" mode, while only the Robert Brown element is coloured.

Colouring is also used in the grammar view to denote what types have been instantiated. Thus the person type in focus in the grammar view has the "misc-info" node grayed out to indicate that Robert Brown has no misc-info element. Note that if another person element is selected in the document view, the colouring of the gammar icons will change accordingly.

4 Related Work

Many syntax-directed editors for programming languages have been created to assist in authoring programs. Such editors may notify a user as to the next possible action through a menu displaying the allowable operations at the cursor position (Arefi, Hughes and Workman 1990). Large recursive grammars result in large and deep grammar trees. Navigating such deep hierarchical trees appears to be difficult for users (Zaphiris, Shneiderman and Norman 1999). TreeViewer (Wittenbug and Sigman 1997) uses font, shade, size and animated transitions to help navigate such trees.

5 Conclusion

We have proposed an approach for editing structured documents through their grammar. The two innovations described here are (1) showing document element types in the fixed arrangement defined by the grammar and (2) providing visual feedback through icon colouring that highlights the choices a user has already made.

6 Acknowledgments

Mark Sifer was with the IBM Research Lab in Haifa when this work was done. We wish to thank Roni Raab for her help in programming parts of Xeena for Schema, and the rest of the Knowledge Management group for their feedback.

7 References

AREFI F., HUGHES C. E. and WORKMAN D. A. (1990): Automatically generating visual syntax-directed editors, *Communications of the ACM*, 33 (3), 349-360.

WITTENBUG K. and SIGMAN E. (1997): Visual Focusing and Transition Techniques, Proceedings of the IEEE Symposium on Visual Languages, 20-27, IEEE Computer Society Press

ZAPHIRIS P., SHNEIDERMAN B. and NORMAN K. L. (1999): Expandable indexes versus sequential menus for searching hierarchies on the world wide web, 99-15,http://www.cs.umd.edu/hcil/pubs/tech-reports.shtml.