The Universal XML Organizer: UXO

Jan-Torsten Milde, Markus Reinsch

Computational linguistics and text-technology group, Fakultät für Linguistik und Literaturwissenschaft, Universität Bielefeld Bertelsmann Lexikonverlag email: milde@coli.uni-bielefeld.de, Markus.Reinsch@bertelsmann.de

Abstract

The integrated editor UXO is the result of ongoing research and development of the text-technology group at Bielefeld. Being a full featured XML-based editing system, it also allows to combine the structured annotated data with information imported from relational databases by integrating a JDBC interface. The mapping processes between different levels of annotation can be programmed either by the integrated scheme interpreter, or by extending the functionality of UXO using the predefined Java API.

1. System architecture

The integrated editor UXO is the result of ongoing research and development of the text-technology group at Bielefeld. The system is an integrated XML-based text editor, which is configurable to a large degree and can so be easily adapted to specific user needs. The editor has been implemented in Java making it possible to execute it on a large number of platforms.

UXO allows to enter the data either by typing in text or by working directly on the displayed structure tree. The structure can be validated by starting the inbuild XML parser (XML4J, IBM). In contrast to standard XML tools available, UXO offers an integrated interface to relational databases in combination with a build-in interpreter for the scheme programming language, used as the systems scripting language. This combination allows to configure the editor for a wide range of varying applications simply by defining control/configuration As these scripts are external to editor, scripts. reconfiguration does not mean to recompile the system. More experienced programmers are able to extend the permanent features of UXO using the powerful API. In principle this API allows to integrate any Java functionality available (see also Reinsch and Milde (2000), Milde (1999)).

The editor manages the full Unicode character set (Unicode, 1996). Its graphical user interface can be completely reconfigured allowing to localize the software and define appropriate control key sequences.

Figure 1 shows the basic system architecture. Internally all data is handled as a DOM instance. It is possible to map database requests (via JDBC, see Klute (1998)), servlet requests (via HTTP) and XML documents onto this model. The editor allows to modify the content of a document and to validate its structure.

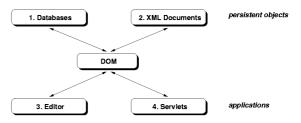


Figure 1: The basic architecture of the system: DOM is used as a central data structure. Database requests, servlet requests and XML-annotated documents are mapped onto a DOM which can then be edited using the UXO editor. Mapping rules are defined using the buildin scripting language (scheme). Alternatively UXO can be extended using the Java API.

2. User Interface

UXO is equipped with a GUI (see figure 2) which is comparable to the ones used in standard XML tools. The main window is split into two content frames displaying a tree structured view and a linear view of the document content. The third frame is used to display status messages. A menu and a button bar allow to access the most important functions of the editor. Other function are activated via control key sequences. The complete GUI (menu entries, messages boxes, control keys, button bar, icons etc.) can be reconfigured by modifying a single external property file. The syntax of the property follows standard JAVA convention and is human readable.

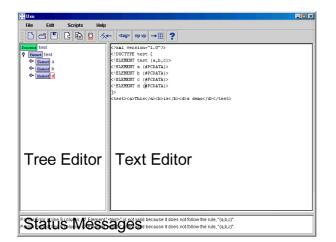


Figure 2: The main window of UXO. The upper left frame displays a tree structured view of the document, the upper right frame displays the XML-annotated linear version. The user is able to enter and modify the document in both views. The lower frame displays status messages (e.g. database requests performed by the system).

All interactions with UXO are guided by the graphical user interface. Dialog boxes, list boxes etc. are defined to help the user concentrate on her work, rather then coping with the difficulty of controlling the system. The user is presented a standard linear version of the loaded text as well as a tree view, allowing to navigate in a very efficient way even in large texts.

3. Database Access

Two main approaches can be distinguished for combining SGML and database technology: extensions of existing query languages and specially designed query languages. Extensions of existing query languages have been developed within a number of projects intended to integrate textual components into classical DBMS: Blake (1994) extends standard SQL to access SGML documents, Christophides (1994) proposes an extension of O2SQL, Yan (1994) presents OSQL and Volz (1996) applies a loose coupling of existing DBMS and Information Retrieval Systems, Le Maitre et.al. (1998) describe SgmlQL, integrating SQL calls into standard SGML-documents. One of the most well known specially designed query language is HyQ, the HyTime query language (see DeRose 1994).

Designing an extension of a query languages or designing new query languages from scratch has the disadvantage of loosing the compatibility to well defined standards. In UXO a different approach is taken: bidirectional functions encoded in a standard programming language (scheme) map between the relational data base entries and the tree structured XML-annotated document instances. So, instead of extending the SGML/XML syntax, we have integrated SQL into the underlying scripting language.

1.1 Scripts

Using scripts is a central feature in UXO. We have decided to incorporate a scheme interpreter into the

system.

The used scheme interpreter (see Bothner (1998a), Bothner (1998b)) is entirely written in Java and makes it possible to define very compact and powerful scripts. It also implements part of the DSSSL syntax, allowing to process SGML/XML documents. Scripts are used in two ways in UXO:

1. As an *editor script*, allowing to extend/configure the basic functionality of the editor (e.g. integrating database information, define complex keyboard macros, extending/adapting the GUI to specific application need etc.):

0	(replace-current-	-word
1	(string-append	"xxx"
2		(get-current-word)
3		"ууу"))

Script 1: A simple Script which replaces (line 1) the word at the current cursor position (lines 1-3) with the concatenation of this word and some new characters.

The example script 1 is a simple script, which simply takes part of the content of the document (getcurrent-word), prepends "xxx" and appends "yyy" to it. Please note that the current word is identified by the caret position, independent from the view mode (linear or tree) currently chosen. A set of about 100 different functions have been implemented, allowing to manipulate the editors content, to request information about the content, to modify the user interaction, to convert different data types, to debug the scrips, to generate XML markup, to transform DOM objects and finally to perform database transactions (see Reinsch (2000)).

Editor scripts are used in UXO in the same way they are used in comparable editors (e.g. Emacs). They simply define configurable macros that can be connected to specific keystrokes or certain elements of the graphical user interface. Editor scripts can also be used to extend the GUI of the Editor. Standard dialog boxes, message boxes or input masks can be defined without any knowledge of the underlying Swing library. More complex extension are also possible, if the integrated Java API is used. In the current test application this has been shown by integrating part of the new Java-Quicktime library into UXO. This allows to display and control Quicktime movies, getting the current timecode, jumping to specific pictures of the movie etc, making it possible to produce transcriptions of the displayed scenes.

2. As an *DSSSL script*, allowing to transform the structure of the XML document (e.g. to convert the content, to perform structured search/replace function etc.):

- 0 (define (update node)
- 1 (send-sql-command (string-append
- 2 "UPDATE "
- 3 (get-resource-string "table")
- 4 " SET "

```
(get-resource-string "set") " =
5
6
      '"(qi node)"'"
7
      " WHERE "
8
      (get-resource-string "update")
9
      " =
10
      '" (data node) "'")
11
      "jdbc:odbc:lilidependenz"))
    (element (update (current-node)))
12
```

Script 2: A more complex script which transfers the content of a given XML-document into the relational database.

Using scripts, the mapping of relational table based data to tree structured XML data becomes relatively easy and very flexible. As you can see in example script 2 sending an SQL command to the database engine is done using the function (send-sql-command command databasename). The command is a simple string containg the SQL-call. In this example data from the XML-structure is used in the SQL-command by calling (gi node) and (data node), accessing the node name and the node content. Once the SQL-transaction is finished, the current node of the XML-structure will be updated to display the new data.

The integrated JDBC interface to relational databases is not restricted to using local databases (such as MS Access on a PC). It is network-based and can connect to most modern relational databases available today.

We hope that this feature will support linguist when linguistic data gathered during field work has to be transferred into the central database used in laboratory work.

Scripts are loaded at runtime of the system, thus they can be edited and reloaded without stopping the editor, making it easier to debug them. Unfortuneatly, if errors occur, the presented error messages are quite unclear. This is mainly due to the fact, that errors most often will lead to internal Java exception, which will be displayed to the user. The Java stack trace does not give very much information about the *scheme* function, which caused the error.

Another way of accessing the functionality of a script is by *compiling* it with KAWA scheme system. KAWA is able to produce Java byte-code, which can then be directly loaded by the Java virtual machine. Such scripts will run at much higher speed. For application developers it will be possible to design and implement scripts, configuring the editor in the desired way and deliver a binary version of their work.

4. Applications

UXO primarily has been designed for building linguistic applications, mainly to support field linguists while describing some language. Still the system is not restricted to linguistic applications. In a number cooperation projects with Bertelsmann and Boellhoff¹, we followed a similar approach as in UXO allowing to semiautomatically generate SGML-annotated product catalogues. At this time (1995-1997) no appropriate integrated working environment for SGML existed, hence we started developing UXO.

In the current version scripts for building up and using a lexicon have been build in. The lexicon will be stored in a relational database, allowing to work with very large lexicons. Once a text has been entered into UXO, calling the lookup-script will add XML-markup to all words found in the lexicon. The user is able to define which information encoded in the lexicon will be used for markup (default is to markup the words category). If unknown words occur in the text, these will be inserted into the lexicon by calling an update-script. If the user inserts appropriate markup, this information will also be inserted into the lexicon. Both the lookup and the updatescript are able to cope with ambiguities.

Using Scheme as the underlying scripting language easily allows to implement standard (computational) linguistic procedures. In the current prototype of UXO a simple dependency parser (DP) for a subset of German has been integrated. The user enters a sentence, which will be parsed by the DP. The dependency structure will be transformed into XML markup and the content of the editor will be changed appropriately:

0	<sentence></sentence>	
1	<n <="" genus="m" idrefs="" td=""></n>	
2	ID="1">Peter	
3	<v <="" idrefs="1 3" td=""></v>	
4	Tempus="present"	
5	ID"2">loves	
6	<n <="" genus="f" idrefs="" td=""></n>	
7	ID="3">Mary	
8		

Figure 3: Parsing the sentence "Peter loves Mary" results in XML-Markup defining the structure of the sentence as well as establishing the dependency relations between parts of the input. The relations are encoded using the ID/IDREFS mechanism of XML:

The DP first generates a list of all words entered into the editor. For each word a lexicon lookup via JDBC-calls will be performed. A single word w is parsed in three steps: first look for all words, which are dependent on w and establish the dependency relations, then look for all words, which govern w and establish the relation and at last store the current word w for further processing. Once the relations have been computed, the result will be transformed into XML-markup of the input text. The relations are encoded using the XML ID/IDREFS

¹ Boellhoff is a manufacturer for connection technology, mainly for the car industry. With approximately 60.000 parts the product range of Boellhoff is rather large. As a result the catalogue has a complexity of a medium size lexicon.

mechanism. Each constituent of the input is assigned a number stored as its ID, allowing to encode the dependency relation by simply listing the IDs of the dependent constituents. In the example (figure 3) the verb 'loves' (IDREFS="1 3") governs the nouns 'Peter' (ID="1") and 'Mary' (ID="3").

5. Conclusion

The Universal XML Organizer UXO is a very well suited basis for the development of tools for XMLstructured data entry and a comfortable tool for complex conversion of and extended search in XML-annotated data. Due to its implementation in Java the system is portable to a number of operating systems without losing functionality. As the system can be reconfigured and extended to a wide degree, adaptation to specific user needs is easily achieved.

References

- Blake, G. E., M. P. Consens, P. Kilpeläinen P-A. Larson, T. Snider, and F. W. Tompa. 1994. Text / Relational Database Management Systems: Harmonizing SQL and SGML. In Proceedings of ADB'94
- Bothner, Per. 1998a. *Kawa: Compiling Scheme to Java* In Proceeding of the Lisp Users Conference, Berkeley, CA.
- Bother, Per. 1998b. *Kawa Compiling Dynamic Languages to the Java VM*. In Proceedings of the Usenix, New Orleans.
- DeRose S. J. and D. G. Durand. 1994, *Making Hypermedia Work. A User's Guide to HyTime*. Kluwer Academic Publishers.
- Christophides, V., S. Abiteboul, S. Cluet, and M. Scholl. 1994. From Structured Documents to Novel Query Facilities. In Proceedings of ACMSIGMOD'94, 313-24. Minneapolis.
- Klute, Rainer. 1998. JDBC ind der Praxix. Addison-Wesley Longman. Bonn.
- Le Maittre, J., Murisasco, E. and Rolbert, M.: From Annotated Corpora to Databases: the SgmlQL Language. In J. Nerbonne: Linguistic Databases, CSLI Publications, 1998.
- Milde, Jan-Torsten. 1999. Effizientes Document Engineering sprachlicher Daten. In H. Lobin, *Text im digitalen Medium*, Westdeutscher Verlag, 1999
- Reinsch, Markus. 2000. Entwicklung eines XML-Tools in Java, Universität Bielefeld, 2000
- Reinsch, Markus, and Milde, Jan-Torsten. 2000. *The universal XML-Organizer UXO*, In Proceedings of PAP2000, Manchester, To Appear.
- Unicode (1996). The Unicode Consortium: *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press.
- Volz M., K. Abere, and K. Böhm 1996. *Applying a flexible OODBMs-IRS-coupling to Structured document handling*. In Proceedings of the 12h International Conference on data engineering (ICDE), New Orleans, USA.
- Yan, T. W, and J. Annevelink. 1994. Integrating a Structured- Text Retrieval System with an Object-Oriented Database System. In Proceedings of

the 20th Conference on Very Large Data Bases (VLDB'94), 740-9. Santiago, Chile.