

***eXitor*: A TOOL FOR THE ASSISTED EDITION OF XML DOCUMENTS**

V. FRESNO-FERNÁNDEZ, S. MONTALVO-HERRANZ, J. PÉREZ-IGLESIAS AND J. Á. VELÁZQUEZ-ITURBIDE

Departamento de Informática, Estadística y Telemática
Escuela Superior de Ciencias Experimentales y Tecnología
Universidad Rey Juan Carlos
C/ Tulipán s/n
28933 Móstoles
Madrid, Spain

e-mail: v.fresno@escet.urjc.es; smontalvo@escet.urjc.es; joperez@escet.urjc.es; a.velazquez@escet.urjc.es

XML and its associated technologies are increasingly being used in Electronic Publishing. This fact is due to some of its inherent characteristics: (1) XML allows separating the contents and the presentation format. (2) XML allows giving a logical structure to information. (3) XML allows associating semantics to the logical structure. There is a general agreement that these features will be very important for the development of the semantic web. In this context there is a demand for XML editors. A comparative analysis of current XML editors reveals that most of them have two general lacks. Firstly, using these editors requires deep knowledge of the XML syntax and its DTD or the XML-Schema particular vocabulary. Their interfaces do not guide to user in the edition process, e.g. by showing him/her the different possibilities available at any moment with respect to vocabulary. This drawback is especially discouraging for naïve users. Secondly, the validation of a document is made on request to specific commands or on saving it. In the worst case, these tools allow generating an invalid document that accumulates errors and overwhelms the user during their correction. In this paper, we describe an editor of XML documents, called *eXitor*, that shares the common features of the main XML editors, but tries to solve the two problems quoted above. With respect to the visibility of XML syntax, the editor supports both naïve and expert users. For the first kind of users, the user can understand the document structure without the need to see the XML source code. For the expert users, the tool allows displaying XML syntax. With respect to the validation problem, the user interface does not allow generating non-valid XML documents. During the edition process, elements and attributes can be inserted guaranteeing that the XML document is well-formed and valid with respect to a given DTD. (In the specific case of XML Schema this tool only supports DocBook XML Schema V4.1.2.3.) In summary, its main characteristics are: (1) The tool can hide syntactic information (XML syntax and DTD or Schema syntax), only displaying the textual contents of the elements. (2) The tool can display syntax information to the advanced user. (3) The document structure is always shown in a tree format. (4) The tool forces to introduce all the compulsory components. (5) At any moment, the document is well formed and valid.

Keywords: XML editor; XML assisted edition; always valid XML documents; DTD; XML-Schema

INTRODUCTION

Nowadays, the Extensible Markup Language (XML) and their technologies are increasingly used in electronic publication. XML is a subset of SGML that has been designed for ease of implementation and for interoperability with both SGML and HTML [1], while preserving universal scope of application. Internet represents the greatest source of information in the world and more and more XML technologies are being used in the Web [2][3].

XML is an extensible language and therefore allows defining markup languages to structure any sort of information [4]. XML source code only contains information about contents and structure, whereas information about appearance is externally kept in XSL or XSLT. In XSL [5] we can find all kind of typographic information: vocabulary, letter sizes, types, etc. Due to this fact, the XML document's contents is independent from its visualization, and therefore the same information can be shown with different appearances. XSLT [6], which is a language for transforming XML documents into other XML documents, allows transforming the XML documents to other formats. These technologies include an XML vocabulary for specifying formatting.

XML allows defining the structure of a document with a previously defined Document Type Declaration (DTD) [7] or an XML-Schema [8]. The main difference between these two approaches is that a DTD has its own syntax whereas the Schemas share the XML syntax. In any case, the previously defined structure for the contents of any document is not mandatory and therefore an XML document only is well-formed when it is checked that it adheres to the language syntax. In the second case, an XML document is valid with respect to a specific schema when its contents is structured according to the schema definition.

Another fundamental characteristic is that XML allows assigning semantics to the logical structures previously defined in the schemas. Once a logical structure has been defined, each element can be matched with a semantics or the complete document type declaration can be matched with a particular ontology. Thus, structured information can be associated in general to a specific meaning structure. This feature makes of XML

“the language of the present and future” and is the base of the Semantic Web. The Semantic Web aims at adding a machine-tractable, re-purposeable layer to complement the existing hypertextual web. In order to realise this vision, the creation of semantic annotation, the linking of web pages to ontologies, and the creation, evolution and interrelation of ontologies must become automatic or semi-automatic processes [9][10]. On the other hand and due to the fact that XML is portable, the research in database area is advancing to transform traditional databases into XML databases and even to transform the relational database to this universal format [11][12].

Other scope where XML is becoming very important is education. In many cases, educators create educational courware for study which is based on proprietary and not compatible formats. As a consequence, the student must acquire various tools to visualize such a material in their learning process. In this context, the development of tools for XML edition has become a fundamental issue.

Since the last 15-20 years, many SGML editors have been developed and in the last years the number of XML editors also has increased. However, their context was limited to experts in markup languages, whereas the needs of generating XML documents currently extends to non-expert authors. On the other hand, these tools exhibit similar functionalities and interfaces. As a consequence, developing editors of educational material that is user-friendly and uses a universal, non-proprietary format as XML is very useful.

In this paper, the development of a common and integrated editor of XML is introduced. This work was conducted within a research project whose aim was to develop a editor to the assisted generation of XML contents in the context of PhD studies. The paper is organized as follows. In the second section, a comparison of common XML editors is shown and based on the accomplishment of several quality criteria. In the third section, *eXitor* is introduced and its internal develop is described. Finally, the Results, the Conclusions and Future Work are presented.

XML EDITORS: A COMPARISON

We have carried out a comparative analysis of eleven XML editors. The comparison was performed mainly attending to different requirements with respect to user interface and edition that a semi-structured editor should satisfy [13].

Firstly, let us see the interface requirements:

- (I1) *Showing a default presentation of the XML document*: The contents of the XML document are showed in a non-tags view. It can be presented by means of style sheets.
- (I2) *Hiding unnecessary structures*: It is shown the structures strictly necessary for edition. The presentation of the document structure can be made by means of a tree schema, tags or any format that assists the users in the creation process.
- (I3) *Viewing the document structure while the document is being created*: The hierarchical structure of the XML document is showed anytime.
- (I4) *Having menus to select only valid structures*: Having different interfaces to allow selecting those elements whose insertion guarantees that the XML document were valid against the used schema.

Secondly, the edition requirements are:

- (E1) *Generating valid documents*: The generated document will be valid with respect to an associated schema. There are two different ways to validate:
 - (ev) *Explicit validation*: The user validates the document invoking an interface command. When this action is performed, the document may have accumulated too many errors to learn from them.
 - (iv) *Implicit validation*: The document is always valid during the creation process in a way transparent to the user.
- (E2) *Guaranteeing the insertion of mandatory elements and attributes*: At any moment, the tool ensures that mandatory elements and attributes are inserted.
- (E3) *Displaying the optional elements at any moment*: For each part of document, only the valid optional elements are displayed.
- (E4) *Creating only valid XML documents*: The editor only generates well-formed and valid documents.

The most outstanding editors were selected and classified into commercial editors, free software or GPL (General Public License), and editors that can be adapted to edit educational material. XML, DTD and CSS documents were created to carry out the comparison and to better identify the differences between the editors. The tools selected for this comparison, classified according to the previous criterion are:

- (a) Commercial editors: *TurboXML* [14], *XMLSpy* [15] and *Xmetal* [16].
- (b) GPL editors: *Amaya* [17], *HTML-KIT* [18], *Oxygen* [19] and *XMLEditPro* [20], *XMLOperator* [21].
- (c) Editors that can be adapted to edit educational support material: *Morphon* [22], *XML-Editor* [23], and *XMLMind Composer* [24].

COMPARATIVE ANALYSIS

Before studying the results of our analysis, it is important to remark great differences in the final aim of the editors analyzed. Some editors are aimed at managing large projects and therefore they not only allow creating XML documents, but also other types of structured documents such as FO (Formatting Objects), WML (Wireless Markup Languages) or RDF (Resource Description Framework). In this first group we can find the tools *XMLSpy* (developed by Altova) or *XMetal* (developed by Corel). These tools also allow creating documents based on other technologies, such as JSP (JavaServer Page) and TSD (Tamino Schema Definition). Other tools such as *Amaya* (developed by W3C) are however aimed at minor projects and therefore allow editing and validating HTML or XML documents and documents with formats as MathML and SGV. In this case, these tools only support the publishing process but it is impossible to create new XML documents.

Let us see the degree of accomplishment of requirements by the analyzed editors:

- (I1) *Showing a default presentation of the XML document*: This requirement is accomplished by all of the editors but *Course Composer*, that uses a particular template for a subset of the *DocBook DTD*, and *XMLOperator*, that only allows showing the structure in a tree view. The rest of the tools have one or more default presentations, the most common being plain text without tags and the result of applying style sheets.
- (I2) *Hiding unnecessary structures*: This requirement is satisfied by all of the editors but *Amaya* and *HTML-KIT*. For the remaining editors, the most common way to show document structure is as a tree view. This tree can be contracted or expanded to show or hide different parts of the document. Other editors as *TurboXML* graphically display each XML element in a representation by boxes where the expansions and contractions also are permitted.
- (I3) *Viewing the document structure while the document is being created*: This requirement is accomplished by all the tools except by *Amaya* and *HTML-KIT*. For the rest, the structure can be displayed during creation in a tree view, graphically or by tagged text.
- (I4) *Having menus to select only valid structures*: This requirement is only accomplished by *Morphon* and *XMLOperator*. In both cases, a menu for selection of elements is presented, where only the elements that ensure the validity can be inserted. The rest of tools allow inserting elements that build invalid documents.
- (E1) *Generating valid documents*: Only *Course Composer* and *Morphon* accomplish this requirement in a implicit way. *Composer* does not allow inserting elements that will lead to invalid documents and notifies it. *Morphon* only shows those elements whose insertion leads to a valid document. *XMLOperator* includes implicit validation, but depending on the complexity of the schema, this validation can be done incorrectly. The rest of the tools, excepting *Amaya* and *HTML-Kit* that are short of any kind of validation, generate valid documents when the user invokes an explicit validation. Only some of them notify their validity before saving documents.
- (E2) *Guaranteeing the insertion of mandatory elements and attributes*: This requirement is directly related to *E1*, since the mandatory insertion of elements and attributes can only be assured with implicit validation. Therefore, this requirement is only accomplished by *Course Composer* and *Morphon*.
- (E3) *Displaying the optional elements at any moment*: None of the analyzed editors accomplishes this requirement. In many cases and depending on the schemas, inserting an element may automatically produce the insertion of more elements and so whole of optional elements are not shown.
- (E4) *Creating only valid XML documents*: This requirement is accomplished by *Course Composer*, *Morphon*, *XMLEditPro* and *XMLMind*. In general, it is accomplished by those tools that build a tree for the document to be created, ensuring well-formed structures. Besides, some of them allow editing the source code to change it, thus in many cases it is uncertain whether the final document will be valid.

Given this analysis, we conclude that the most remarking editors are *Morphon* and *Course Comoser* because they satisfy a larger number of requirements and they are the only tools that generate valid documents with respect to XML-Schema.

COMPARATIVE RESULTS

The results of the previous comparison are summarized in the Table 1. The accomplishment in the case of *eXitor*, that will be introduced in the next section, is showed too in this table. Moreover, after the analysis of requirements, other more general conclusions could be extracted. Respect to the tools with an explicit validation, the use of these editors require a widespread knowledge of the XML syntax and the particular vocabularies. The interfaces do not guide to the user in the creation process and so, do not show the different possibilities with respect to the vocabulary that can be selected in any time. Thus, due to the validation process is explicit, in many

cases these tools will allow accumulate many errors and will carry problems to the user being inexpert in the language syntax. These lacks will be very important if we do not assume a widespread knowledge for the most of XML editors users. As it was advanced, the social and technologic reality brings this situation. Respect to the tools with an implicit validation, could be concluded that these editors improve the lacks of the tools with explicit validation, however, none of them perform all the required requirements and specially the concerned to displaying the whole the optional elements in any moment (E3).

TABLE 1 – Summary of the accomplishments of each one of the analysed editors, for the interface and edition requirements.

	Interface				Edition			
	I1	I2	I3	I4	E1	E2	E3	E4
<i>Amaya</i>								
<i>Composer</i>					(iv)			
<i>HTML-KIT</i>								
<i>Morphon</i>					(iv)(ev)			
<i>Oxygen</i>					(ev)			
<i>TurboXML</i>					(ev)			
<i>Xmetal</i>					(ev)			
<i>XMLEditPro</i>					(ev)			
<i>XMLMind</i>					(ev)			
<i>XMLOperator</i>					(iv)			
<i>XMLSpy</i>					(ev)			
<i>eXitor</i>					(iv)			

Requirements related to interface (I) and edition (E): (I1) Showing a presentation by defect of the XML document; (I2) Hiding unnecessary structures; (I3) Viewing the document structure while the document is being created; (I4) Having menus to select only valid structures; (E1) Generating valid documents; (E2) Guaranteeing insertion of obligatory element and attributes; (E3) Displaying whole the optional elements in all the moment; (E4) Only XML documents with valid structure are created.

***eXitor* : A NEW APPROACH IN ASSISTED EDITION OF XML DOCUMENTS**

Once the selected XML editors has been analyzed and finding that none of them satisfy the totality of the required requirements, emerged the needs of developing a new tool that resume all of them and would be centered in an educational context. In this section, a short explanation about this develop is introduced. This work is supported by the research project TIC2002-01387 of the Spanish Research Agency CICYT and by the Grant for PhD studies AFC2001-0544-IP of the Ministerio de Educación, Cultura y Deporte.

This development is located into the educational innovation project AFC2001-0544-IP supported by the Ministerio de Educación y Cultura, whose aim is the edition of didactic material in XML format. Great emphasis was placed in the usability in the university environment.

The main goal of *eXitor* is to guide user in the process of edition and creation of XML documents, offering in any moment all possible elements and attributes to insert. Displaying whole the optional elements in any moment is a requirement not satisfied by none of the analysed tools and it is achieved in the proposed approach. Thereby, any user could edit electronic notes, e-books or any XML document in a simple way and without the necessity of a widespread knowledge of the language. All the proposed requirements in the previous section, related to user interface and edition, are performed by *eXitor* and specially, the total implicit validation is achieved. We consider the implicit validation as base in assisted edition process into an educational environment because the grammatical errors cannot appear and so, they will be not accumulated. To reach this validation three steps must be follow: 1) inserting always the obligatory elements and attributes when it corresponds; 2) must being possible to select all the optional elements in any moment and any document position; 3) selecting only those elements that imply, in each moment, a valid document against a given schema.

To validate a document, a schema must be considered and there are two possibilities to define content models, the valid order and nesting of elements: Document Type Declaration (DTD) and XML-Schema. The XML DTD is a language derived by SGML. It can be used to define content models and the datatypes of attributes of elements within a document. However, it has some limitations. First, they are written in a different syntax and they have no support for namespaces. Second, only offer limited datatyping (0, 1 or infinite). The XML-Schema have XML syntax and offers a range of new features over the DTD. It has richer datatypes: booleans, numbers, dates and times, URIs, integers, decimal numbers, real numbers, intervals of time, etc. It is possible creating other types and aggregate types and it support attribute grouping that allows common attributes that apply to all elements in a schema to be explicitly assigned as a group. Other new features are the inheritance

and namespace support that allows the co-existence of multiple schemas without name conflicts between those schemas. Thus, a content model defined by a DTD is “closed”, it describes only what may appear in the content of the element. XML Schema admit two other possibilities: “open” and “refinable”.

Because of their advantages, the XML-Schema trends to be imposed and the use of DTD decrease gradually. But the reality today is other, there are more grammars written in DTD syntax than in XML syntax and so, we considered important supporting indistinctly DTD and XML-Schema to design *eXitor*. In the case of DTD, other reason is that supporting DTD is being supported any SGML document. Thereby, our editor give support to any grammar in the case of the DTD and, at this moment, do not support in a generic way the XML-Schemas and only documents with the *Docbook XML Schema V4.1.2.3* can be generated. Docbook [25] is a DTD mainly designed to create bibliographic material in XML format and *XML Schema V4.1.2.3* was, during the development of *eXitor*, a beta version XML-Schema of the corresponding DTD. The aim of the project, the assisted edition of XML documents in an educational environment, was the reason to implement this version of the Docbook as a first approach to support generically the XML-Schemas. Futures efforts will be focus to give a complete support to XML-Schemas.

DEVELOPMENT OF *eXitor*

In this section, the different steps to carry out the development of the XML editor *eXitor* are introduced. The tool was developed in JAVA, being a multiplatform language and so, portable. It was selected because is an Object-Oriented language that offers a great number of classes with varied functionalities and moreover it could be considered as a mature technology, proved adequately and accepted. Great number of projects all around the world are developed in this language and the Internet is its natural space. The development of the editor is divided in several phases. The system architecture is showed in the Figure 1.

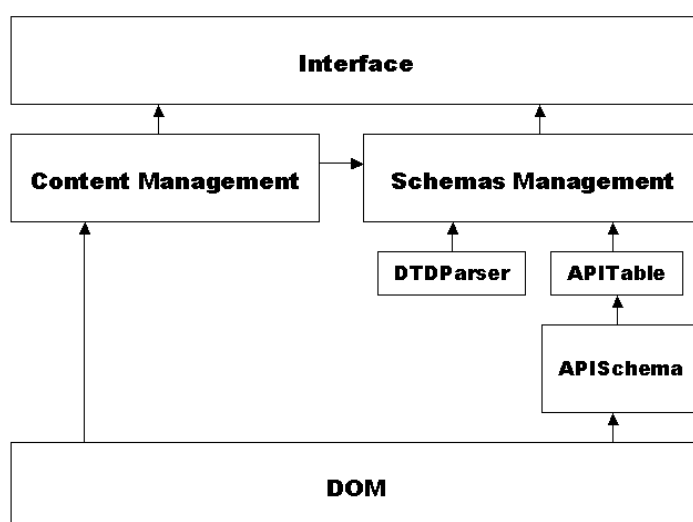


FIGURE 1 – *eXitor*'s architecture

In a first phase, handling the schemas -managing the information about the structure of the document- is needed. The documents structure can be stored in a DTD or a XML-Schema and in each case the treatment is different. To access to the information in the DTD a library called DTDParser [26], programmed in JAVA, was used. This API is designed as a set of classes that allow to analyse the DTD grammar and retrieving any needed information in any time. To access to the included information in the XML-Schema, not any library was found available, and so, two class hierarchies were designed and developed in JAVA: APISchema and APITable. The APISchema allows to retrieve all the information needed to generate a XML document (elements, attributes, entities, etc..). It is a set of classes built over the DOM API, that will be introduced later, to extract the information from the XML-Schema and to store it in dictionary structures in memory. The second set of classes, the APITable, allows to retrieve the information from the dictionary, in an ordered and well structured way, for later use in superior levels into the application.. These two hierarchies give a similar functionality for XML-Schema that the DTDParser for DTD.

Storing and managing the information from the schemas is needed to accomplish the implicit validation while the user is generating the document. For this task, the Xerces classes (Apache)[27] were used because regular expressions are implemented. Depends of the development of the document in a specific moment and where the user would want to modify -inserting, deleting or modifying- an element, a pattern matching process

with the possible document after the changes and the schema is arrived. Only those elements from the schema that give a positive matching will be show to the user. In this way the optional elements would be showed, a element would be deleted, inserted, etc. These regular expressions are used too in the case of attributes and to decide the automatic insertion of mandatory elements. This treatment with regular expressions carries to the accomplishment of the *I4*, *E1*, *E2* and *E3*.

In a second phase, a persistent structure in memory to store the document and accessing to the contents is needed. With this views, the most adequate is DOM (Document Object Model), where a XML document is represented as an object that belongs to a Document class. The DOM approach is based on storing in memory the structure (a XML documents always has a tree structure) and the contents. A set of methods to access to this information of the document, the possibility to validate against a associated schema and rebuild the document, since the image in memory, are some of the features offered by DOM API. In *eXitor*, all modifications in the structure or in the content of the document are achieved with this representation.

The last phase in the development of the interface is the design, which is split in two parts: a *tree view*, to maintain in a visible way the structure, and a *text view*, to edit the textual content. In the left side, the document is represented in a tree view mode, where each XML element is a tree node. Only across the user modifications in the structure or contents of the document could be achieved and so, a always-valid structure is forced and the requirement *E4* is carried out. While the document is being created, their structure is always showed and the requirement *I3* is achieved too. By means of contextual menus in each one of the nodes, the different options to insert, to modify and to delete the elements are allowed. Each one of these operations have an associated interface and across of them, all the contents of the elements and attributes are inserted. Selecting a specific attribute, or the value for an attribute, could be done by means of these interfaces. Expanding and contracting the tree structure to visualize the most interesting parts in each moment is other functionality of *eXitor* and furthermore with it, the requirement *I2* is accomplished. The text view mode is located in the right side of the interface and show the contents of the elements, hiding the XML tags, proper characteristic of this language. Then, the user visualizes only the contents in a plain text format. This is the presentation by defect in (*I1*). The user could solicit moreover a non-editable version of the XML source code.

For the synchrony between the contents of the elements in DOM and the presentation by defect, a Content Manager module was developed to associate the positions of the contents of each element from tree view, in the left side, to the content visualization window in the right side. Similarly, selecting a element in the tree, its content and the content of its children is selected too, whose suppose a help when the user is editing the content of a node. The interface was completed with several bottoms associated with the most useful functions.

RESULTS

The development of *eXitor* was fundamentally focused in the accomplishment of whole the proposed requisites in the comparative analysis, to be a complete semi-structured document editor. All of these criteria were covered during the development. However, the emphasis was on the treatment of the schemas to ensure the implicit validation, because the main goal in this project was that any user would create XML documents in a simple and guided way, being experts or naïve users. Thus, contributing to increase the document creation in those users that edit documents and do not have a widespread knowledge of the XML syntax neither the specific schema.

Achieving the implicit validation in a strict way, will carry automatic processes as are element insertions or element selections. In many cases, and depending of the specific grammar, inserting (or deleting) a specific element implies the needs to insert (or delete) other elements to obtain a final document being valid against the schema. This property is not found in the rest of the analysed tools and it is a fundamental issue in a assisted edition process. In the development of *eXitor*, the implicit validation was considered more important than other issues. The educational approach forces an additional effort to access, and to store in structures in memory, all information related to the valid order and nesting of elements. datatypes and values of attributes of elements within a document are retrieved too and so, all information over grammar is extracted and it could be used in the guided edition process.

The interface of *eXitor* guides in this way to user in the edition process, making impossible to realize any action that implies a non-valid document. However, this interface is not deperated at the last stage although it covers all the basic actions in edition process and accomplish the whole of the user interface requirements. In Figure 2 we can observe several application's screenshots and different interfaces for the elements and attributes insertion.

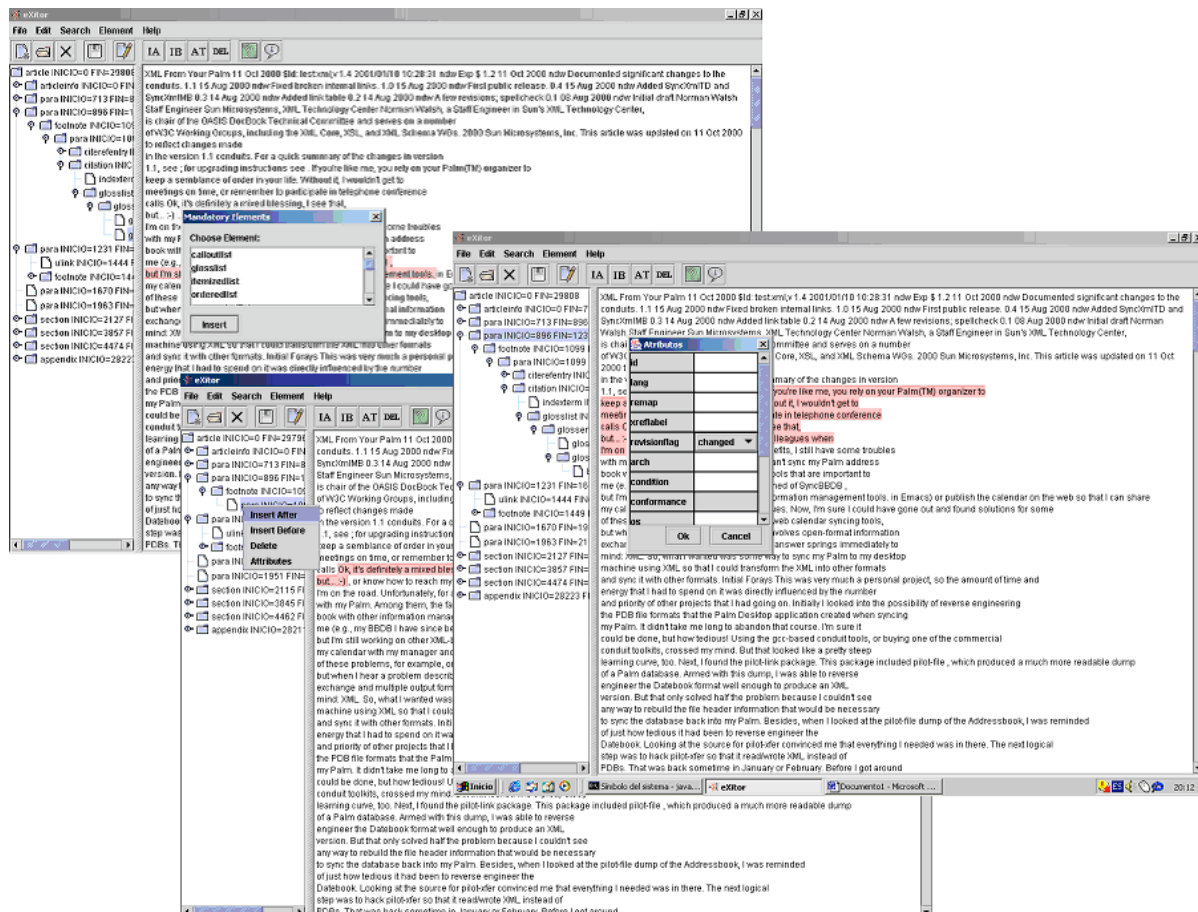


FIGURE 2 –eXitor's screenshots.

CONCLUSIONS AND FUTURE WORK

In this paper, the development of a new XML editor, called *eXitor*, has been introduced. Nowadays and mainly due to the Web, the XML document generation and the growing use of tools to edit these type of contents is been increased. XML is not an easy language to learn, and thus it is not reasonable to demand XML experts to edit the whole of XML documentation, therefore, XML editors should hide syntax and structure details. A set of XML editors was selected to perform a comparative analysis to find the accomplishment of the main features related to edition and user interface. After the analysis was reached, none of the selected editors accomplished all the proposed requirements and especially those relatives to a guided edition.

eXitor is a XML editor developed into an educational environment and focused on guided edition. The proposed editor accomplish all required requirements and perform correctly implicit validation. In this way, the document is always-valid during the edition process. This feature is performed by other editors but in no case, all the optional elements are showed to be inserted. In those cases, only a subset of these elements is offered to user to insert. Our approach guarantee that all the optional elements are showed always. Thus, an insertion automatic process of mandatory elements is reached.

With regard to the future we will focus on the development of a more advanced and friendly interface. We will carry out a XML-Schema complete support, in the same way that DTD is supported. We will include some templates that make easy the edition of those more common document types and a transformation module, that lets to convert XML document to other formats as could be PDF, RTF, HTML, etc. Through this improvements, *eXitor* could be a widely known tool and be useful for any type of users.

ACKNOWLEDGMENTS

This work is supported by the research project TIC2002-01387 of the Spanish Research Agency CICYT and by the Grant for PhD studies AFC2001-0544-IP of the Ministerio de Educación, Cultura y Deporte.

REFERENCES

1. *Extensible Markup Language (XML) 1.0 (Second Edition)*. (<http://www.w3c.org/XML/>)

2. Beckett,DJ .*Connecting XML, RDF and Web Technologies for Representing Knowledge on the Semantic Web*, XML Europe 2002, IDEAlliance, 2002
3. Heflin, J. and Hendler, J. *Semantic Interoperability on the Web*. In Proceedings of Extreme Markup Languages 2000. Graphic Communications Association, 2000. pp. 111-120.
4. Zisman, A. *An overview of XML*. Computing & Control Engineering Journal, 11 (4). IEEE. 2000
5. *The Extensible Stylesheet Language (XSL)*. (<http://www.w3.org/Style/XSL/>)
6. *XSL Transformations (XSLT)Version 1.0*. (<http://www.w3.org/TR/xslt>)
7. Logical structures - DTD. (<http://www.w3.org/TR/REC-xml>)
8. *XML-Schema 1.1*. (<http://www.w3.org/XML/Schema>)
9. Hendler, James, Berners-Lee, Tim and Miller, Eric. *Integrating Applications on the Semantic Web*. Journal of the Institute of Electrical Engineers of Japan, Vol 122(10), October, 2002, pp. 676-680.
10. Heflin, J. and Hendler, J. *Dynamic Ontologies on the Web*. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*. AAAI/MIT Press, Menlo Park, CA, 2000. pp. 443-449.
11. J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, B. Reinwald. *Efficiently Publishing Relational Data as XML Documents*. VLDB Journal 10(2-3), 2001.
12. Albrecht Schmidt, Martin Kersten and Menzo Windhouwer. *Querying XML Documents Made Easy: Nearest Concept Queries*. 17th International Conference on Data Engineering. 2001.
13. Wilkinson et al. *Document Computing. Technologies to Managing electronic Document Collections*. Kluwer Academic Publishers.
14. *TurboXML* .<http://www.tibco.com>
15. *XMLSpy* (<http://www.xmlspy.com>)
16. *Xmetal* (<http://www.softquad.com/products/xmetal/>)
17. *Amaya* (<http://www.w3c.org/amaya>)
18. *HTML-KIT* (<http://www.chami.com/html-kit>)
19. *Oxygen* (<http://oxygen.sync.ro/>)
20. *XMLEditPro* (<http://www.daveswebsite.com/>)
21. *XMLOperator* (<http://www.xmloperator.net/>)
22. *Morphon XML-Editor* (<http://www.morphon.com/>)
23. *XMLMind* (<http://www.xmlmind.com/xmleditor>)
24. *XMLMind Composer* (<http://www.myrnham.co.uk/>)
25. Norman Walsh and Leonard Muellner. *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. 1999.
26. *DTDParser*. (<http://www.wutka.com/dtdparser.html>)
27. *Xerces2 Java Parser*. <http://xml.apache.org/>