Editing EML 1.1 Preliminary survey of existing tools

Educational Technology Expertise Centre OTEC
Open University of the Netherlands

Editing EML 1.1

Preliminary survey of existing tools

COLOPHON

Title:	Editing EML 1.1 Preliminary survey of existing tools
Author(s):	Francis Brouns
Project:	WP 2, Development Programme
Project manager:	ing H. Vogten
Project members:	E.J. van den Berg dr.ir. F.M.R. Brouns drs. H.J.H. Hermans H.G.H Martens ir. G.W. van der Vegt ing. M. Verhooren ing. H.F. Vogten
Programme:	Research and Development into Learning Technologies
Programme chair:	Prof.dr. E.J.R. Koper
Programme assistant:	Mieke Haemers
Report type:	Report
Publication date:	14 February 2003
Distribution:	OTEC

Educational Technology Expertise Centre (OTEC)
Open University of the Netherlands

Editing EML 1.1
Preliminary survey of existing tools

OTEC report series

The Open University of the Netherlands develops higher distance education and is a central partner in a consortium for the renewal of higher education. Educational technological innovation is one of the main fields of interest. The educational and educational technological expertise of the Open University of the Netherlands are bundled in the Educational technology expertise centre (OTEC). This centre is involved with tasks concerning the development, innovation, research and evaluation of the Open University and its consortium partners. These tasks are performed in close collaboration with directorates and faculties of the Open University and/or its consortium partners.

OTEC publishes a report series. This report is part of that series.

Development Programme reports and some documents can be obtained from:

Open University of the Netherlands
Secretary Development Programme P.O. Box 2960
6401 DL Heerlen
the Netherlands
Tel. +31 45 5762624
Fax. +31 45 5762800

Internet: http://www.ou.nl/otec

Table of contents

1. Introduction

The Development Programme of OTEC, Open Universiteit Nederland has developed 'Educational Modelling Language' (EML). EML is a semantic notation for education (units of learning). It can be used to describe didactical models and related processes.

The main implementation of EML is an XML (eXtensible Markup Language) application in the form of a XML DTD (Document Type Definition).

The Open Universiteit Nederland has used EML 1.0 for some time now to create units of learning. Courses specified in EML 1.0 are delivered through the Edubox (version 2.0.6) environment. The authoring environment currently in use is based on an SGML editor, Framemaker+SGML 5.5.6, because there were no suitable XML editors available at the time.

Past experiences and the participation in the IMS Learning Design working group have resulted in the adaptation of EML 1.0 into EML 1.1. EML 1.1 takes advantage of new XML constructs and is available only as XML binding. The major binding for EML 1.1 is a XML DTD.

The members of the Development Programme needed to get acquainted with the new version of EML (1.1) and test the binding. Additionally, the revised EML 1.1 was the basis for the new Edubox environment being developed by Perot Systems on behalf of the Open Universiteit Nederland. Therefore it was imperative that the team members of the Development Programme gained extensive knowledge of the new EML and were able to produce test material for the Perot Edubox player. The team indicated that a text editor would not suffice and that a more sophisticated editor would be needed. An initial attempt to configure the existing authoring environment showed that Framemaker+SGML 5.5.6 would not suffice and indicated the need to explore XML editors.

This document describes a short survey of available XML editors in their ability to create and edit EML 1.1 files. The editor was intended solely for use by the members of the Development Programme. The assumptions were that no changes were made to the EML 1.1 DTD and that the editor would be used 'off-the-shelf', so without modification. However their functionality in an authoring environment was also considered.

The report is intended for the members of the Technology project of the Development Programme. The survey was only preliminary and not at all intended to perform a full evaluation of requirements and functionalities of editors.

Nor does this report specify any requirements for an authoring environment. When EML 1.1 is used in a production environment where authors and academic staff are using it to create educational material, a much more complex and sophisticated authoring environment is needed. Another project was foreseen to specify requirements for such an environment.

2. Changes in EML 1.1

Although EML 1.0 is an XML application, the Open Universiteit Nederland is still using an SGML version of EML. At the time the project started and EML was being developed only a few XML editors existed. As XML is a derivative of SGML, it was no problem to create an SGML DTD and still be fully compliant with the XML application. The XML constructs used in EML 1.1 however, are very hard to implement in an SGML DTD or SGML editor. Therefore an evaluation of XML editors to replace the current SGML editor was in place.

The new version of EML 1.1 contained some semantic changes. The major change was the complete removal of structured content from the design. Instead the design contains references to resources. EML no longer provides constructs to structure content. The only elements remaining are those needed to create dynamic content and structures that make use of results/outcomes of constructs in the learning design.

Another change is the incorporation of the IMS content package and metadata and the possibility to replace certain structures (like metadata and expressions) with other schemas. This requires the use of new XML constructs like namespaces (explained later).

So, EML 1.1 actually consists of 2 bindings. The major binding is the one that specifies the unit of learning and learning design. The second binding provides the dynamic elements. These elements can be used to create dynamic content (also referred to as emlcontent) by combining them with other vocabularies, in this case XHTML. These dynamic elements are also referred to as EML global elements.
In addition two included schemas are defined: one for the metadata structure and one for the expression elements.

Although EML actually consists of 2 bindings and a definition of two included schemas, for practical reasons they are all maintained in one file. This file was created and maintained in Near&Far Designer, the tool in use by the Development Programme. Later on it will be explained why this imposed some restrictions.

The new EML specification allows substitution of certain schemas, but the EML 1.1 binding has provided a fixed set. In future specifications, such as the IMS Learning Design, these schemas might not be fixed. Although this was not a major issue in the current survey, its impact was taken into consideration.

3. Terminology and conventions

EML 1.1 is used to specify a unit of learning and an instructional design. No binding, except the global elements, is provided for content. Content (i.e. actual text presented to users) is provided in resources. EML 1.1 provides a mechanism to link the resources to the design. These resources can consist of so-called webcontent, i.e. those contents that can be rendered or handled by a web-browser, including binary documents, like Word documents etc. The other type of resource is the so-called emlcontent. Emlcontent consist of a XHTML document combined with the EML global elements. XHTML is HTML in XML notation.

In the remainder of this document the term EML is used in several contexts. It can refer to the EML 1.1 binding (DTD) or documents conforming to the EML 1.1 binding (DTD). It is also used as synonym for unit of learning or learning design, unless explicitly stated otherwise. The context should be sufficient to determine the meaning. EML refers to the latest version, EML 1.1, unless stated otherwise.

The term EML is never used for content; emlcontent is used to refer to content in resources that make use of the global elements.

Tagnames are shown in italic between angular brackets, like <*unit-of-learning*>. Attribute names are in italic. Examples are provided in Courier typeface, like `xsi:schemaLocation`.

4. Aims and constraints

The editor would serve multiple purposes. It could be used to create EML and emlcontent files to familiarise project members with the new EML version, and to create test material for the new Edubox implementation. The first aim imposed fewer restrictions on the editor than the second did. It would be less important that the generated files completely met all specifications needed by Edubox, or be fully XML compliant. It might even be acceptable to adapt the files manually before testing them in Edubox, when the testset would remain limited in size.
However, the survey was also intended to determine whether the editors were suitable for producing fully XML and EML compliant files, which did not need any processing afterwards.

Another constraint applied to the survey. The Open Universiteit already had considerable educational material coded in EML 1.0. This material would be converted to EML 1.1 (and from SGML to XML). The editor preferably should be able to read and edit the files generated by the converter.

The only acknowledged binding for EML 1.1 was a XML DTD. No schema was created yet, except for an automatically generated schema that did not take inheritance and import of namespaces into account. The assumption was made that it should be possible to create EML and emlcontent by using the EML 1.1 DTD. It would be acceptable to produce an adapted version of the DTD as long as it entailed minor changes. This assumption proved wrong, as explained later.

It should also be possible to use the editor with EML without resorting to extensive adaptation of the editor, except for the normal configuration needed, like compiling a DTD or creating a basic template.

4.1 Namespaces

As indicated earlier, EML 1.1 already combines elements from different bindings. This does not have to be a problem, when EML 1.1 is only processed by a dedicated application under the control of the Open Universiteit. However, it always has been the intention that others would use EML, which is already the case. EML 1.1 is also set up to allow substitution of elements by other schemas and is the basis of standardisation activities in the IMS Learning Design Working Group. Because of this, it becomes important to be able to distinguish the schemas to which certain elements belong. It is

even more important when there is a naming conflict as is the case for emlcontent, where there are duplicate names for XHTML elements and certain EML elements. XML provides a mechanism for this: namespaces. Use of namespaces enables distinction of elements, prevents naming conflicts, enables context dependent processing etc.

A XML namespace is a collection of names in XML documents used as elements and attributes. The namespace is identified by an URI reference. The W3C recommendation on namespaces provides a method to associate elements and attributes with the namespace.

Namespaces are declared by the use of reserved attributes. Declaring the namespace by the use of the *xmlns* attribute is also referred to as default namespacing.

Example of default namespacing

```
<product xmlns="http://www.example.org/product">
  <table>
    <model>….</model>
  </table>
</product>
```

In this example the namespace has been declared on <*product*> and thus applies to all elements within <*product*>.

Another mechanism is to declare a namespace prefix (using *xmlns*:) and prefixing all relevant elements. The prefix can be declared at the root element of the XML document, or at the relevant element. All elements in the namespace need to receive the prefix. Advantage of declaring the prefix at the root element is that the prefix needs to be declared only once.

```
<prod:product xmlns:prod="http://www.example.org/product">
  <prod:table>
    <prod:model>….</prod:model>
  </prod:table>
</prod:product>
```

The namespace applies to the element in which the namespace is declared, and all its descendants (inherits down/inwards), until overridden. Multiple namespaces are allowed.

```
<product xmlns="http://www.example.org/product">
  <table>
    <model>
      <html:p xmlns:html="http://www.w3.org/1999/xhtml">A square …
      </html:p>
    </model>
  </table>
</product>
```

In this way, the namespace prefix needs to be declared at every instance of an html element. The html namespace prefix could have been declared on the root element instead of at the relevant element. Then the namespace has to be declared only once and can be used on all relevant HTML elements.

Attributes are not necessarily in the namespace that applies to the elements, unless explicitly placed in the namespace. The authoring process becomes rather complex

when attributes are placed in the namespace, because those attributes need to be prefixed. No default namespacing is possible with attributes.

Example where the elements are but none of the attributes is placed in the namespace:

```
<product xmlns="http://www.example.org/product" id="p1" name="oak">
  <table>
    <model>
      <html:p xmlns:html="http://www.w3.org/1999/xhtml">A square …
      </html:p>
    </model>
  </table>
</product>
```

Example where the attribute 'id' is placed in the namespace, but the 'name' attribute not:

```
<product xmlns="http://www.example.org/product"
xmlns:prod="http://www.example.org" prod:id="p1" name="oak">
  <table>
    <model>
      <html:p xmlns:html="http://www.w3.org/1999/xhtml">A square …
      </html:p>
    </model>
  </table>
</product>
```

Namespaces can be used in a DTD, but this is rather complicated. It is even more complex to combine different namespace in a DTD. According to W3C specs (Modularization of XHTML) it is possible when designing a DTD in a modular way. This is a rather complex method, for which no tooling was available in the project. That would make the creation and maintenance of the DTD very cumbersome. The DTD editor Near&Far Designer in use by the project is not suited for this. Whether XML editors are able to parse such modular DTDs was unknown. Therefore this approach was not taken. Instead an alternative was tried.

Namespaces in a DTD can be indicated by the use of the *xmlns* attribute, or by naming the element including the prefix. This prefix then is not seen as the namespace prefix, but as part of the XML name. However this can result in an unnecessary complex authoring environment, depending on the editor used. Editors that are not namespace aware will allow the use of the *xmlns* attribute as normal attribute, depending on the author to provide the value for the attribute. Namespace aware editors however, will handle *xmlns* attributes in a special manner; often preventing it's use as 'normal' attribute.
Additional problems arise with default namespacing, because the namespace inherits onto descendants until overridden. This means that default namespacing (in a DTD) can only be used when the other vocabulary (in our case XHTML) has provided the *xmlns* attribute for all elements. Several parsers (like MSXML) do not allow default namespacing for documents based on a DTD.

For testing purposes, a DTD for emlcontent was created, which used prefixes for the EML global elements; addition of the *xmlns* attribute to all elements was seen as a too large deviation from EML 1.1. The document instance requires a default namespace

declaration for the XHTML elements and a prefix declaration for the prefixed EML elements, and requires the prefix for all relevant elements.

```
<html xmlns="http://www.w3.org/1999/xhmtl" xmlns:eml="http://eml.ou.nl/eml11">
  <head>
    <title></title>
  </head>
  <body>
    <p>Some text here</p>
    <eml:view-property/>
    <p>Some more lines </p>
  </body>
</html>
```

However, as the EML elements could contain elements from other namespaces, or the XHTML namespace, there must be a mechanism to reset or override the namespace for descendants.

```
<html xmlns="http://www.w3.org/1999/xhmtl" xmlns:eml="http://eml.ou.nl/eml11">
  <head>
    <title></title>
  </head>
  <body>
    <p>Some text here</p>
    <eml:special>
      <p xmlns="http://www.w3.org/1999/xhtml">This is a special</p>
    </eml:special>
    <p>Some more lines </p>
  </body>
</html>
```

In the above example the namespace has to be declared again for the element *<p>* inside the EML global element *<special>*. The DTD does not provide this, unless the *xmlns* attribute is added to all elements in all namespaces, or prefixes are used for all namespaces.

Not all editors are 'namespace aware'. Even namespace aware editors often still depend on the author to set the namespace declaration on the relevant elements.

4.2 Converter issues

The converter is based on XSLT. The parser used for the converter (MXSML) was not capable of validating documents against a DTD, requiring a schema instead. This added additional constraints to the editor, because document instances generated by the converter contained additional attributes to specify the schema.

The unit of learning documents that are generated by the converter are validated against a XML schema. XML Spy 4.3 was used to convert the DTD into a XML schema. Namespaces on all elements except *<unit-of-learning>* were lost, because this was a simple DTD to schema conversion. This was acceptable for Edubox.

Because all unit of learning documents are validated against a schema, the following 2 attributes (*xmlns:xsi* and *xsi:schemaLocation*) are added to the top-level element:

```
<unit-of-learning
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://eml.ou.nl/eml11 eml11-XMLschema-
   generated.xsd">.
```

These attributes are specific for the use of schemas and do not occur with DTDs. This complicated the process, as these instructions needed to be added to document instances when they are published in Edubox.

In EML 1.0 structured content existed. In EML 1.1 content is no longer provided in the unit of learning document, but in external resources. The converter extracts all content from EML 1.0 documents and creates resources. These resources consist of XHTML combined with EML global elements where necessary. All the EML elements are prefixed. There are different namespaces and thus prefixes for <*metadata*> and the other EML global elements. Default namespacing is used for the XHTML elements. The prefix and namespace declarations are added to the root element and the relevant EML elements, the default namespace for XHTML is added to every descendant of body. The converter is not yet able to validate these documents. So the *XMLSchema-instance* and *schemaLocation* declarations are not added to emlcontent documents.

The editor should be capable of declaring the prefixes and namespaces at the root element and at the respective children/descendants.

4.3 Valid or well-formed EML

A unit of learning should be a valid EML and a valid XML document. A valid XML document should conform to a DTD or a XML schema. A valid XML document usually contains a public or system identifier in a doctype declaration (`<!DOCTYPE html PUBLIC "-//OUNL//DTD XHTML emlcontent//EN" "emlcontent.dtd"]>`) when using a DTD or a *XMLSchema-instance* plus *schemaLocation* processing instruction (`<html xmlns="http://www.w3.org/1999/xhtml" xmlns:eml="http://eml.ou.nl/eml11" xmlns:emlmd="http://eml.ou.nl/eml11/metadata">`) when validating against a schema.

EML 1.1 is specified as a XML DTD. Namespaces are indicated by the use of the *xmlns* attribute at the relevant elements. This original DTD has been used to create unit of learning documents and edit documents generated by the converter.

Documents presenting emlcontent should at least be well formed. However, the authoring process is much easier when the document is based on a DTD or schema, because it guides the author in constructing the EML global elements and also the XHTML document. So, also for emlcontent the test was conducted with a DTD or schema, in order to create well-formed and valid documents.

The use of both the doctype declaration and the schemalocation attributes is not advisable. The doctype declaration can only point to a DTD, while the schemalocation points to a schema. This might confuse implementations.

Most editors will provide an absolute system path or an absolute file URI. This presents interoperability problems, complicating re-use. It also presents problems with

those validating parsers that expect a correct URI. The XML schema standard specifies that a URI should be used to point to the schema location, not a file path.

5. Choice of editors

This survey was not intended as a full-scale investigation of all available editors. A few criteria were taken into account when selecting editors. The editor should support namespaces, preferably support XML schema, provide a WYSIWYG interface. Possibilities to adapt and configure the editor via scripting and APIs were considered an advantage, although for this survey the editor was used 'off-the-shelf'. A demonstration version of the editor should be available, because there was no funding for this survey.

A quick Internet search was conducted, and several editors were downloaded. Some of them were rejected, because either the interface was too crude, or they failed to read the EML DTD.

As Framemaker+SGML was already in use in the current authoring environment, it was included in the testing.
Some experience with XMetaL was gained in a previous test and therefore XMetaL was included in the test.
Two new editors were added: XML Spy and TurboXML. TurboXML was added at a later stage and was only tested for emlcontent.

Most available editors are intended for experienced users or users wanting to learn or work with XML. These kinds of users are familiar with XML concepts. The target audience for EML is not likely to be familiar with those concepts, nor would there be any need to. For these authors it would be important to use an editor that would allow the user to work in a manner they are used to. A WYSIWYG interface and minimal training are among the required features. These aspects were taken into account while evaluating the editors, but were not leading. It was less important for the current survey, because it was intended for experienced users familiar to XML.

6. Editing a unit of learning

As indicated above, it was considered acceptable to use only the EML namespace and omit other namespaces when creating a unit of learning document for Edubox. Based on this assumption, it might be possible to use the existing authoring environment. Two problems existed with the current authoring environment: it works only with a DTD and it can only import SGML no XML, although XML export is possible.

The first problem, use of a DTD, is no real problem. The document instance remains the same when creating SGML or XML except for the first few lines of the document, which optionally contains a XML declaration. Most editors insert the XML declaration (`<?xml version="1.0" ?>`) when creating XML files; some only provide a doctype declaration. This is allowed because the XML declaration is not mandatory.

The main problem is the addition of the *xmlns:xsi* and *xsi:schemaLocation* attributes to the root element. These attributes are not present in the DTD. They either need to be added to the DTD, or to the internal subset in the document instance. In either

case, this presents interoperability problems. In some editors this might even result in conflicts, as they will recognise the xsi parameters, but not in combination with a DTD. No tests have been done where the attributes have been added either to the DTD or to the internal subset. This is rather complicated, because most editors do not recognise the prefixed attribute name as valid.

Other problems lie in the *xmlns* attributes. These attributes are present in the DTD as #FIXED or read-only attributes. Most XML editors will not insert these attributes into the document instance, unless the author manually inserts the attribute (actually resulting in default namespacing for the relevant elements). Several editors/parsers fail to validate document instances that contained both the namespace declaration at the root and this default namespace declaration at respective elements.

6.1 Framemaker+SGML5.5.6 as unit of learning editor

Framemaker+SGML 5.5.6 is a product of Adobe (http://www.adobe.com). It is a product mainly used by technical writers to produce and format complex documents, both structured (SGML) and unstructured.

Advantages

Framemaker+SGML5.5.6 (referred to as FM) is currently in use by the Open Universiteit in the authoring environment.

It is a multi-platform application.

FM has a graphical WYSIWYG interface, with extensive formatting capabilities.

FM is an SGML editor, but is capable of exporting to XML.

Authors and developers are familiar with the product.

Adaptation of the product is possible in several ways; some require programming via an API, others rely on creation of templates, or configuration files.

Support is provided by Adobe, but the large developer and user community provides more effective and efficient support.

Disadvantages

Although FM can export to XML, FM can not import XML files, so round tripping was not possible.

Another major problem is that Framemaker+SGML5.5.6 does not validate the document when exporting to XML. FM provides a validation mechanism for the document while in FM format. This validation is not as strict as needed for SGML (or XML). For example, FM allows a less strict format for the value of ID attributes than is allowed in SGML. A second, stricter SGML validation is applied when exporting to SGML. As the second validation is not applied when exporting to XML, this might result in invalid EML files.

FM creates the XML declaration, but omits the doctype declaration (because it creates only well formed XML) and omits the xsi parameters. This is a disadvantage, because the document can no longer be validated automatically. These documents need additional processing after FM before they can be used in Edubox or other editors.

Because FM creates only well-formed XML documents, it will insert all attributes with default or fixed value into the document instances. This results in elements with the *xmlns* attribute present in the instance, i.e. a default namespace declaration. (Most other editors will omit these attributes, because they assume the presence of the DTD). The validator used for the converter, which is based on a schema, flags this as errors, because the *xmlns* attribute is not allowed on descendants. This is probably a bug in the validator. It is likely to present problems in Edubox as well.

Finally, FM creates a XML stylesheet processing instruction and the corresponding CSS stylesheet. This might impose problems when publishing in Edubox, because it directs Edubox to use this CSS stylesheet. The author might not have intended to supply a stylesheet, although Edubox should support it. So, this XML stylesheet processing instruction (PI) should be removed. As FM does not provide a XML API, documents have to be processed by an external application to remove the stylesheet PI.

Documents can only be created and edited as SGML documents. Therefore the use of prefixes is prohibited; the colon is not allowed in SGML element-names.

As FM does not import XML documents, a much stricter version control and management procedures need to be in place.

Another option was explored, in which FM was only used to create SGML files. These were then converted to XML using either XMetaL version 2.1 or 3.0. Both editors however, failed to properly read the SGML documents created by FM, due to bugs in the XMetaL implementation.

Conclusion

In short, although FM produces documents that are EML valid, the resulting XML files need post-processing, before being Edubox compliant. Framemaker supplies a standard API and an SGML API, but not for XML events. Any post-processing therefore need to occur outside Framemaker, using additional tooling.

6.2 XMetaL as unit of learning editor

Corel recently acquired XMetaL (http://www.xmetal.com) (originally from Softquad).

XMetaL is a SGML/XML editor that comes in several flavours. Functionality partly depends on operating system used (Win9x vs. NT, Win2000).

Two versions have been evaluated. XMetaL 2.1 can edit documents based on a DTD. XMetaL 3 should also support XML Schema.

Advantages

According to the specifications, XMetaL provides many scripting possibilities. There was little prior experience with XMetaL. Adaptations and configuration would require time to explore these possibilities.

XMetaL can be used without any prior configuration by advanced authors. It then relies on the author to indicate the DTD to base the document on. XMetaL then automatically compiles this DTD in a proprietary format. It is also possible to distribute only the compiled DTD. This prevents changes to the DTD. However, XMetaL is rather unstable, even with compiled DTDs, in that it does not always read the compiled DTD properly.

XMetaL provides multiple views of a XML document, amongst which a graphical WYSIWYG interface. This graphical interface can be configured via XMetaL specific instructions in combination with CSS stylesheet instructions. This interface is not very intuitive.

Disadvantages

Although XMetaL should support SGML, the problems are such that XMetaL for SGML is not advisable.

XMetaL is available for Windows only. Unicode support depends on Windows version.

When XMetaL is not configured, authors need to indicate the DTD on which the document must be based. This assumes that authors are familiar with XML concepts, which usually is not the case.

The graphical interface is hard to navigate and not intuitive.

It is very hard to navigate in documents and edit without using the mouse. This is a disadvantage, because it increases rsi problems.

XMetaL generates a doctype declaration with a system identifier that is an absolute file path. When a template is provided a relative path can be specified, but it remains a file path, instead of an URI. This restricts interoperability.

Implied attributes with default or fixed value are not entered into the document instance, unless explicitly entered by the author. This applies to the *xmlns* attribute as well, thus relying on the author to insert this attribute at the proper places to declare the correct namespace.

It is not possible to indicate the required order of elements in complex content models. The author needs extensive knowledge of the DTD.

XMetaL 2.1 is not namespace aware.

XMetaL 3 supports XML Schema partly. It does not support import of other namespaces. Any schema for EML requires import of other namespaces, in particular for emlcontent.

XMetaL is unstable and seems to 'forget' configurations.

XMetaL is unable to open a document instance where the required additional attributes have been added to the root element.

XMetaL 3 can not add the namespace declaration to the root element, because the generated schema does not provide a target namespace, or a default namespace. The schema would need adaptation.

Support is available during the first 30 days after purchase. Additional support can be purchased. Since the survey, XMetaL has been taken over by Corel. No information about quality of support is gathered.

Conclusion

Maybe it would be possible to configure XMetaL via scripting so that it could be used to create EML 1.1 documents. Due to its instability, the first impression is against XMetaL.

6.3 XML Spy as unit of learning editor

XML Spy is a product of Altova (http://www.altova.com). The test was conducted with XML Spy 4.3 IDE.

The XML Spy 4 IDE is intended for developers. It is more than an XML editor, providing much functionality like DTD and schema creation, and XSLT.

There is also a document editor, intended for authors. This client however needs to be configured extensively, requiring XSLT scripts to be developed in the IDE, before the client can be used, and is available as plug-in only. This client was not evaluated.

Since the test XML Spy 4.4 and XML Spy 5 has been released. These new releases contained several bug fixes, but are still unable to provide solutions for the problems encountered. In particular the latest release seemed to re-introduce previous bugs and introduce new ones.
The new release also contained a separate document editor. This document editor still relies on extensive configuration and XSLT scripting, which needs to be created in another application.

Advantages

According to the specification, XML Spy can be scripted and adapted.

XML Spy can be used to create XML documents without configuration, when the author indicates the DTD or schema to be used. This is an advantage for experienced authors, but not for the intended target audience of EML users.

Response of support, at least during the first 30 days of the license, is quick.
Resolution of problems might not be as quick.

Disadvantages

There was no prior experience with XML Spy, so time had to be spent to explore the application.

XML Spy is a suite combining much functionality, targeted mainly to developers. This seemed to compromise quality and stability.

XML Spy is a XML editor, not able to read SGML files.

XML Spy is only available for the Windows platform.

XML Spy IDE does not have a WYSIWYG interface. A WYSIWYG interface can be created for the document editor (separate tool) by creating a XSLT template. This is rather cumbersome because EML is very complex.

Document instances can be created in XML Spy IDE in the 'enhanced grid view'. This is a tabular view. This view is very limited, because the fields to enter elements and text are limited in size. Navigation in this grid view is not intuitive and relies heavily on use of the mouse.

Editing requires the use of the mouse. It is very hard to impossible to enter text, elements or attributes without the use of the mouse.

The interface is not intuitive. The author need to indicate whether elements/attributes need to be appended, inserted, or are children. This assumes extensive knowledge of the DTD/schema.

XML Spy has a very strict validation, when explicitly calling the validation tool, but not while creating and editing documents. While editing, invalid documents are easily created. XML Spy does not warn the author about this. Errors are reported only when explicitly validating. In addition, XML Spy does not provide a full overview of all validation errors, but stops the validation at the first error. It is not possible to skip some errors and validate the rest of the document. There is no indication whether it is the only validation error, or that other errors are present.

The strict validation prevents the use of XML Spy by educational designers. They knowingly create (temporarily) invalid templates as basis for authors. It is not possible for them to check the remainder of the template.

Implied attributes with fixed or default values are not entered into the document instance.

Required attributes are automatically inserted with empty string "". This empty string remains in the document until the author provides a value. The validation does not always reports validation errors on this, because empty strings are allowed for some attributes.

XML Spy generates a doctype declaration with a system identifier that is an absolute file path. This prevents interoperability.

The elements are not listed in the correct order. Authors need extensive knowledge of the DTD/schema to know which element to insert.

The same problems, inserting the required additional schema attributes to the root element, exist as reported with the other editors, when basing documents on a DTD.

Functionalities of XML Spy seem to vary with the view to such extend that what is functioning in one view is not available or reporting errors in another view.

Support is free during the first 30 days of purchase. After this period, support is available only when a support and maintenance contract has been bought.
New release and upgrades are available only under the support and maintenance contract, but not under the regular license.

Conclusion

Again the basic problem was the addition of the schema attributes to the root element.

The other aspects seem to preclude its use for beginning authors and EML users.

6.4 TurboXML

TurboXML is a product of Tibco (http://www.tibco.com). It is a Java-based multi-platform IDE for XML, providing facilities for creation of XML schemas, XML documents and DTDs.

TurboXML 2.3.1 has been tested.

Advantages

TurboXML provides the user with multiple views. Although the graphical view is not really WYSIWYG, this view is more intuitive than the enhanced grid view of XML Spy.

The graphical view can be customised.

TurboXML provides a good graphical view of DTD and schema files.

Disadvantages

Turbo XML generated a doctype declaration with absolute file URI when assigning a DTD to the instance. This prevents interoperability.

No information about support could be found.

Conclusion

TurboXML was not really tested in the creation of unit of learning documents. The same problems with the schemalocation attributes exist with TurboXML as with the other editors.

7. Editing emlcontent

Actual content for EML can be of two types: either webcontent or emlcontent. Webcontent is all content that can be handled by a web browser, either natively, via plug-in, or via the shell. Webcontent can consist of native web pages, HTML, XHTML or XML, text, or can consist of binary documents, etc.
Emlcontent consists of the combination of the EML global elements with another vocabulary. The most obvious choice for this is XHTML. To create emlcontent at least 2 bindings need to be combined. In a DTD this is difficult, XML schema provides mechanisms for this. However, there is not yet a schema for XHTML.

Another need for the use of namespaces, when combining the EML global elements with XHTML, is to prevent a naming conflict for two elements (<*title*> and <*meta*>).

The converter handles the multiple namespaces in the document instances by adding prefixes and namespace declarations to the root and to the required EML elements and a default namespace declaration to HTML elements. The editor should be able to read and edit these documents, i.e. need to be able to handle documents with prefixed elements, default namespacing and multiple nested namespaces.

Any editor would be able to handle documents with nested namespaces. The EML global elements are not only added to XHTML elements, the content of EML global elements in turn contains XHTML elements.

The converter creates only well formed documents, omitting any doctype declaration or schemalocation attributes. Well-formed documents can be opened in nearly all editors. However, no guidance can be offered when editing these documents, because no DTD or schema is specified to validate against. So the editor would need to provide a mechanism to specify a DTD or schema to validate these documents against.

Several DTDs were created to conduct the tests. The basis was the XHTML strict or transitional DTD. The EML global elements were added to the content models of body, all block-level elements and all inline elements.

There were two variants of the DTDs; one without prefixed element names and one with prefixed element names. There were two exceptions to this: elements <*title*> and <*meta*> occur also in XHTML (with differing models) and needed to be prefixed in all DTDs to prevent naming conflicts.

All document instances needed to declare the eml namespace prefix even when the EML global elements were not prefixed, because of the duplicate element names.

A correction of the EML global element <*test*> was required, because the model erroneously contained an endless loop.

7.1 Framemaker+SGML5.5.6 as editor of emlcontent

Framemaker could not be used to create emlcontent with prefixed element names, because Framemaker is an SGML editor. In SGML no colon is allowed in the element name.

The use of default namespacing was not possible without changing the DTD by adding a *xmlns* attribute to all elements, including the XHTML elements. It would also require the author to manually insert the *xmlns* attributes in all elements.

Framemaker 7 was acquired after the test. Framemaker 7 is capable of editing XML documents, based on a DTD. According to the specification, there is no support for XML Schema, but it supports namespaces. Additional tests would be required to investigate its suitability as EML editor.

7.2 XMetaL as editor of emlcontent

Because of the duplicate names, namespaces had to be declared in the instance. When creating new documents, these either had to be added to the template, or had to be added manually afterwards. Upon editing, XMetaL considered these as attributes not present in the DTD and reported an error. Consequently, the instance could not be validated and only be opened as well formed. For well-formed documents there is no guidance about allowed elements and documents.

When the DTD to be used was indicated (manually), many problems remained. XMetaL failed to indicate all allowable elements: sometimes only the elements already present in the document were shown, sometimes only some of the EML global elements.

XMetaL added a doctype declaration to all documents. This might be a problem for other implementations.

Conclusion

Both versions of XMetaL failed to provide the author with the options to create valid emlcontent files.

7.3 XML Spy as editor of emlcontent

XML Spy generated a XML declaration and doctype declaration including an absolute file path when creating new documents based on the DTD. However, it failed to add the *xmlns* attribute to the instance, unless explicitly (manually) added to the element. This is not allowed in combination with prefixed elements; in this case the declaration of the prefix is required, not the default namespace. Default namespacing is also unwanted, because it inherits to descendants.

XML Spy did not add namespace declarations to the document.

XML Spy had many problems with those elements consisting of the ANY content model.

XML Spy could handle nesting the EML global elements in an HTML element, but crashed when those EML elements in turn contained HTML elements. This is required for example in the special element.

Conclusion

No emlcontent documents, except really simple ones, could be created in XML Spy.

7.4 TurboXML as editor of emlcontent

Advantages

TurboXML has a rather intuitive interface.

Preliminary testing was very promising.

Disadvantages

Problems arose with more complex documents, in particular with those elements consisting of the ANY content model and those elements that required nesting of namespaces.

Nesting of namespaces was not handled properly.

There was no guidance on inserting valid elements when namespaces were nested.

Conclusion

Many problems, mainly related to namespaces remained, to such extend that no valid emlcontent instances could be created.

8. Emlcontent with schema

Whenever namespaces are required XML schema is a better choice than a DTD. Several attempts were conducted to create a schema for emlcontent that allowed creation of valid documents. There is no acknowledged schema for XHTML, but a preliminary version was available. In addition the XHTML Strict DTD was converted to a schema with Turbo XML.

Two editors, capable of creating documents based on schemas were used to test the schemas: XML Spy 4.3 and TurboXML 2.3.1.

The schema needed adaptation to TurboXML because it had specific requirements for the XML namespace.

Many problems remained with these schemas. Both editors (XML Spy and TurboXML) failed to create valid instances. There were problems with the elements consisting of the ANY content model, but also with choice groups. Sometimes the problems were restricted to the graphical interface. The textual view sometimes allowed elements to be inserted, which could not be inserted in the graphical interface. However the textual view did not offer any guidance on allowed elements.

Both editors would finally crash or hang when creating emlcontent instances.

9. Findings

None of the available editors could be used, either to create EML or emlcontent. Considerable adaptations would have been required, both of the bindings (DTD or schema) and of the tooling.

In particular the use of namespaces introduced many problems. Then there were still many bugs in available tooling that prevented creation of documents in a graphical view. Working in a textual view could circumvent some of these. However the textual views did not offer any author guidance, thus requiring that authors have extensive knowledge of the binding.

Emlcontent consists of XHTML expanded with the EML global elements. Emlcontent should be well formed XML, but use the eml namespace for the EML global elements, by the use of a prefix or via default namespacing. As the EML global elements in turn can contain XHTML elements, namespaces have also to be used for the XHTML elements when used as descendants. To complicate it even further, the XHTML DTD already uses namespaces as it uses XML attributes as xml:lang and xml:space.

There is also a naming conflict for 2 elements, <*title*> and <*meta*>. These elements are used both in XHTML as in the EML global elements. In both bindings the contentmodel differed: the <*title*> element contains attributes in the XHTML namespace, and not in the eml namespace. The meta element has completely different contentmodels and attributes in both bindings. Because of this the use of namespaces becomes imperative.

Several elements in the EML global elements consist of the ANY content model, implying that any element can occur as descendant, even elements from another namespace. Because of this, any descendants that are not from the eml namespace either have to declare a new namespace or return to the null namespace. In order to do this, the *xmlns* attribute is required. However, the existing XHTML DTD does not specify this attribute. So, even when editors would allow this attribute, it remains a manual action by the author. Another possibility would be to use fully qualified names for every element, which means creating a new XHTML DTD and adding a prefix to all elements.

Although all editors seemed to have problems with the same issues, in addition to aspects of user-friendliness, no common denominator could be found. Most tools claimed to use their own XML parser, but no details were provided.

The test revealed additional problems with the EML 1.1 binding. The EML 1.1 binding is initially created as a XML DTD describing the unit of learning, the external schemas and the global elements. Several of these major constructs have a specified namespace. Some elements are used in each of these major constructs. Because these elements do not contain an explicit namespace (by the use of the *xmlns* attribute), these elements inherit the namespace of the ancestor and are seemingly from different namespaces. This includes langstring, notification, role-ref, email-data, hint, etc.

There is a parameter entity defining a set of elements to create expressions. These elements belong to a separate namespace, because they could be replaced by another set of expression elements. However the namespace has not been specified on each of the individual expression elements and can not be defined on a parameter entity. Instead the namespace is specified on the element <*expression-schema*> and on the element <*expression*>, each containing the parameter entity as its content model. However, the parameter entity containing the expression elements is also used in other instances. In these cases the namespace could not be specified as a default namespace, using the *xmlns* attribute, but it might be possible to declare a prefix for all these elements in the root of the document instance and use prefixes throughout the instance.

Above-mentioned issues in the binding would need to be corrected.

Use of a DTD binding, in particular for emlcontent is prohibited, because:
- will create a doctype declaration instead of a XML PI
- will have to add xmlns:xsi attribute declaration to internal DTD subset in document instance. However most editors can not handle the prefixed attribute name in a DTD.
- will have to create new DTDs, either to add the xmlns attribute to all elements, or to add prefixes to all elements
- use of xmlns attribute will only work in editors, which are not namespace aware. Then it will be an author action to set the attributes, which is not advisable. When this attribute would be fixed, most editors will not write the attribute value into the document instance (which is valid).
- when prefixing all elements, a global namespace declaration is needed in the document instance. The editors do not create this XML PI, instead they create a doctype declaration.

The use of a DTD might be possible for unit of learning documents when Edubox accepts that no namespaces are declared for the EML expression elements, EML <metadata> elements, IMS contentpackage and IMS Metadata elements.

In order to create EML and emlcontent with current tooling a schema needs to be developed that specifies exact models, instead of the many open models currently present in EML.

There are additional problems not raised here that need solving before any of the editors could be applied in an authoring environment. One of the major changes in EML is the extensive cross-referencing mechanism for items and resources. This can not be solved in a simple editor, but requires a sophisticated authoring environment.

10. Conclusions

It soon appeared that none of the major editors were suitable for editing EML and emlcontent. Although some could be made suitable for creating the EML unit of learning (under certain assumptions), none were capable of creating emlcontent files.

Creating a specific binding and implementing closed schemas for those constructions that are now open might prevent some of the problems.

Even when a valid schema could be constructed, many problems remain with the tooling. All tooling would need adaptation, either via configuration or via scripting to pre-process or post-process the files.

Then, there still remain bugs in the current tooling that probably can not be circumvented by additional programming or changes in the binding.

11. Glossary

design	specifies the instructional design of the unit of learning
DTD	Document Type Definition Declarations laying down vocabulary and syntax, or grammar of documents.
EML	Educational Modelling Language A specification to model pedagogies and related processes. http://eml.ou.nl
HTML	Hyper Text Markup Language
SGML	Standard Generalized Markup Language (SGML) A text format to structure documents. A 'meta' language to describe markup languages. A markup language contains a vocabulary and syntax. ISO Standard 8879: 1986
XHTML	Extensible Hyper Text Markup Language A reformulation of HTML 4.0 in XML.
XML	Extensible Markup Language A simple, very flexible text format derived from SGML (ISO 8879). It provides a universal format for structured documents and data. It is a 'meta' language to describe markup languages.
unit-of-learning	root element of EML document, consisting of metadata describing the file, design containing the didactical scenario, resources

12. References

EML. http://eml.ou.nl
HTML. http://www.w3.org/MarkUp/
IMS Global Learning Consortium, Inc. http://www.imsglobal.org
IMS Metadata Specification. http://www.imsglobal.org/metadata/index.cfm
IMS Content Packaging Specification.
http://www.imsglobal.org/content/packaging/index.cfm
Modularization of XHTML. http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/; http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/dtd_module_rules.html#s_dtd_module_rules
Namespaces in XML. World Wide Web Consortium 14-January-1999.
http://www.w3.org/TR/REC-xml-names
RFC2396. IETF (Internet Engineering Task Force) RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998.

SGML. http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387
XHTML. http://www.w3.org/MarkUp/#xhtml1
XML. http://www.w3.org/XML/
XML Schema. http://www.w3.org/XML/Schema
XML Schema Part 0: Primer W3C Recommendation, 2 May 2001 http://www.w3.org/TR/xmlschema-0/
XML Schema Part 1: Structures. W3C Recommendation 2 May 2001. http://www.w3.org/TR/xmlschema-1/
XML Schema Part 2: Datatypes. W3C Recommendation 2 May 2001. http://www.w3.org/TR/xmlschema-2/

OTEC 2002/44