Annotations with EARMARK for arbitrary, overlapping and out-of order markup

Silvio Peroni
Department of Computer Science
University of Bologna
Bologna (Italy)
speroni@cs.unibo.it

Fabio Vitali
Department of Computer Science
University of Bologna
Bologna (Italy)
fabio@cs.unibo.it

ABSTRACT

In this paper we propose a novel approach to markup, called Extreme Annotational RDF Markup (EARMARK), using RDF and OWL to annotate features in text content that cannot be mapped with usual markup languages. EARMARK provides a unifying framework to handle tree-based XML features as well as more complex markup for non-XML scenarios such as overlapping elements, repeated and non-contiguous ranges and structured attributes. EARMARK includes and expands the principles of XML markup, RDFa inline annotations and existing approaches to overlapping markup such as LMNL and TexMecs. EARMARK documents can also be linearized into plain XML by choosing any of a number of strategies to express a tree-based subset of the annotations as an XML structure and fitting in the remaining annotations through a number of "tricks", markup expedients for hierarchical linearization of non-hierarchical features. EARMARK provides a solid platform for providing vocabulary-independent declarative support to advanced document features such as transclusion, overlapping and out-of-order annotations within a conceptually insensitive environment such as XML, and does so by exploiting recent semantic web concepts and languages.

Categories and Subject Descriptors

I.7.2 [**Document And Text Processing**]: Document Preparation – *Markup languages*.

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods – *Representation languages*.

General Terms

Languages

Keywords

Markup, Overlapping Markup, XPointer, Earmark, OWL.

1 INTRODUCTION

EARMARK (Extreme Annotational RDF Markup) is our proposal to integrate in a single conceptual approach both embedded markup (*a la* XML) and external annotations onto resources (*a la* RDF), so as to make the advantages of both approaches available at the same time in any situation in which either one would have been seen as more advisable over the other.

Embedded markup (XML and the like) and external annotations (such as RDF) are both used in current practice to express facts about documents. They are rarely seen as related in aims because their different approaches and syntaxes renders the facts expressed about the document quite differently: XML markup is typically embedded, coexisting with the content being marked up and giving the document a very different appearance. It is therefore useful in "making explicit what is conjectural or implicit, a process of directing the user as to how the content of the text should be (or has been) interpreted" [5]. On the other hand, RDF [10] is typically external and refers to content via the addressing mechanism of URIs, and does not impact on the actual document, and as such is "a language for representing information about resources in the World Wide Web" [12]. Similarly, RDFa [1] is an approach to embed semantic annotations within XML-based markup languages, but does not express facts to any additional structures to the ones identified by the XML markup itself.

These sentences contain the pros and cons of the two approaches:

- XML markup can make assertions on arbitrary fragments of the content of text documents, identifying the string they annotate with the simple device of embedding it within the XML annotation itself. On the other hand, embedding can express exactly one fact about the fragment it is wrapping, and making multiple assertions over the same content fragment in XML is cumbersome and inelegant. Furthermore, assertions cannot overlap nor ignore the order of the content, but need to be organized in a hierarchical fashion and in document order. Finally, assertions over the whole document have no mandated or suggested syntactical approach (they are often expressed as separate sections at the beginning or end of the document, or as attributes within the document, or as sections titled "meta", etc.)
- RDF annotations can make assertions on identifiable resources by using their URIs as the subject of the assertion. RDF can make any number of annotations over the same resource without restrictions on hierarchies, overlap or multiplicity. On the other hand, it needs URIs to refer to resources, i.e., it can only make annotations over content organized in containers addressable by URIs, e.g., either a whole document or those document fragments which, by means of some XML-like embedded markup, are provided with an identifier.

Literature exists on ways to extend one approach beyond their limitations: RDFa [1] and microformats, for instance, were

DocEng '09, September 16-18, 2009, Munich, Germany.

created to allow authors to assert additional facts about XHTML elements above and beyond the semantics implicit in the XHTML element's name (such as a paragraph also being a license or a heading also being an author or a title [2]). Analogously, overlap markup has been proposed to allow for multiple hierarchies of assertions to be associated to the same text fragments or even to partially overlapping fragments [6,13].

Intuitively, although each of these proposals aims at extending the corresponding approach in the right direction, it stops short of providing the advantages of the other approach altogether. This means, in brief, that although RDFa allows one additional RDF assertion to be associated to existing XHTML markup, it can allow neither an arbitrary number of RDF assertions on the same markup, nor any RDF assertions on text fragments that are not wrapped within XHTML markup. Similarly, although overlapping markup proposals such as LMNL [18] and TexMecs [11] do allow multiple assertions on the same text fragment, and even on partially overlapping fragments, they still require document order to be well-defined, cycles in assertions to be avoided, and provide no natural way to make assertions about the whole document.

As mentioned, RDF as a language does provide the most general way to make assertions about a document and its content, as long as an addressing mechanism is used to precisely associate the assertion to the corresponding fragment (i.e., as long as any text fragment can be considered as a URI-identifiable resource). The full XPointer schema [7] of the XPointer W3C standard does in fact provide a way to specify URIs pointing to arbitrary fragments of text, XHTML and XML documents even when they are not wrapped within a markup element. Yet, although the XPointer framework ended up being a W3C Standard [9], the XPointer schema of the framework (the one that is the most useful in our situation) has not been ratified by the W3C, and at the moment chances are underwhelming that it will ever be.

In this paper we describe the Extreme Annotational RDF Markup (EARMARK), a proposal to bring full RDF expressiveness to document markup (or, equivalently, to provide full fragment addressing to RDF assertions). EARMARK does so by describing an ontologically sound model that formalizes the Location and Range concepts of the XPointer schema of XPointer, and a natural way to express embedded markup as RDF assertions and RDF assertions as embedded markup within the document's content. With EARMARK it is possible in a simple way:

- to express arbitrarily complex assertions about text, XHTML and XML documents without restrictions to the overall structure of the assertions, including single hierarchies, multiple hierarchies, direct acyclic graph and generic fully-featured graphs (even cyclic graphs) both according to the document order or independently of it;
- to convert embedded markup into external RDF assertions;
- to express external RDF assertions as embedded markup according to a number of well-known embedding styles (including XML, RDFa, TexMecs and LMNL);
- to express assertions on the whole resource, on elements provided with IDs, and on arbitrary strings using the same approach;
- to express virtual elements, empty elements and structured attributes as introduced by TexMecs and LMNL;
- to express out-of-order and repeated uses of the same text fragments (properties which has been so far not allowed by any embedded markup);
- to verify through the use of OWL ontologies a number of interesting properties over EARMARK documents, including content model validation, pattern verification and consistency of semantic assertions against OWL ontologies.

The paper is organized as follows: first a specific use case is illustrated, a karaoke application where the screen display of the lyrics of the song needs to cater for text repetitions (e.g., the chorus), different choices of lyrics (e.g., depending on the gender of the singer) and additional, partially overlapping, annotations over the lyrics (e.g., times, comments, hypertext links, etc.). The EARMARK model is subsequently introduced by describing the basic ontology and providing a first rendering of the karaoke example. Then the issue of externalization of embedded markup and linearization of RDF annotations are discussed and detailed and several linearization options are discussed. Finally, a surprising formal result is discussed, according to which any choice of RDF annotations can in fact be linearized into an XML, or TexMecs, or LMNL document (even if overlapping, or non directed, or acyclic).

Being a mechanism for non-embedded markup, EARMARK can be also considered as a solid candidate, as advocated by Ted Nelson in [15] for expressing xanalogical annotations on texts and arbitrary documents to create the complex web of transclusions that constitute the heart of Xanadu's proposal [16].

2 A FICTITIOUS USE CASE: KARAOKE

To illustrate the difficulties of embedded markup *a la* XML, let us consider a fictitious karaoke application in which lyrics are displayed on a screen in sync with a voice-less recording of the corresponding song. The lyrics are marked up according to the TEI vocabulary [17] and a number of additional annotations are associated to the text.

Let us consider for instance the song "And I love her" by the Beatles, one of the most famous and sung songs of the history of pop. The lyrics of the original Beatles rendition in the album "A hard day's night" go as shown in Table 1[1].

Table 1. Song structure of "And I Love Her" by The Beatles

Title	*And I love her*
1	*I give her all my love / That's all I do / And if you saw my love / You'd love her too*
Chorus	*I love her*
2	*She gives me ev'rything / And tenderly / The kiss my lover brings / She brings to me*
Chorus	*And I love her*
3	*A love like ours / Could never die / As long as I / Have you near me*
4	*Bright are the stars that shine / Dark is the sky / I know this love of mine / Will never die*
Chorus	*And I love her*
-- Solo --	
4	*Bright are the stars that shine / Dark is the sky / I know this love of mine / Will never die*
Chorus	*And I love her*

[1] as available in any number of lyrics site, e.g., http://www.lyricsfreak.com/b/beatles/and+i+love+her_10026463.html

In the history of pop music, "And I love her" has been one of the most covered songs, and recordings exist of both male and female performers. The first difficulty for our karaoke application is that the lyrics of the song, when performed by a female singer, need to change gender throughout, the title becoming "And I love him" and every instance of the word "her" changed into "him", "she" into "he", etc. Furthermore, although the sequence of stanzas and choruses of the first four stanzas are pretty much similar in all performances, what happens afterwards varies considerably among performers: for instance, we examined the original and 10 different and reasonably famous covers of it.

As shown in the following table, the Beatles repeat the fourth stanza, just like Connie Francis, while Mary Wells repeats the first stanza, Barry Manilow and Pauli Carman repeat both the third and the fourth stanzas, and the chorus is repeated a different number of times in each performance. Even more differently, Patty Pravo did an Italian cover of the song with brand new lyrics that is even structurally different since it is composed of individually different 5-verse stanzas (numbered 5, 6, 7, 8) instead of the 4-verse stanzas followed by a shared 1-verse chorus of the English lyrics. A karaoke application that would play the song according to the orchestration and recording of different performers would then have to consider, in some way, all these differences (Table 2).

Table 2. The structures of stanzas and chorus of 11 different renditions of "And I love her" [2]

Performer	*Gender*	*Structure*
Beatles	F	1 C 2 C 3 4 C s 4 C
K. Lattimore	F	1 C 2 C 3 4 C 4 C C C C
B. Manilow	F	1 C 2 C 3 4 C s 3 4 C C
P. Carman	F	1 C 2 C 3 4 C s C 3 4 C C
Scorpions	F	1 C 2 C
C. Francis	M	1 C 2 C 3 4 C s 4 C
M. Wells	M	1 C 2 C 3 4 C s 1 C
E. Phillips	M	1 C 2 C 3 4 C s 4 C C C C
P. Bennett	M	1 C 2 C 3 4 C
Friends of Distinction	M	1 C 2 C 3 4 C s 4 C X
P. Pravo (Italian)	F	5 6 7 8 s 8

The simplest solution is to provide a different XML document for each performance, making these renditions, in practice, completely different songs. A different approach would be to store the actual text of the lyrics (including the gender differences) and separately some instructions on how to combine them in a sequence that matches the chosen recording. This would have the additional advantage of allowing repeated phrases (e.g. the chorus) to be stored only once.

The disadvantage of this, of course is that the XML syntax does not allow such sequencing instructions in general, and the specific vocabulary (in our case, for instance, TEI) may not provide for them in any natural way. In fact, far from being a straightforward XML document with descriptive markup wrapping the fragments of the text according to their immutable roles, such document would end up as a fairly procedural affair with specific markup that needs to be understood and exploited by a very specific and application dependent tool, hardly the reason for switching to XML in the first place.

Let us add to this a number of additional information such as timing of each lyrics' line, chord annotations for performers, and maybe even a few text-based fun facts about the song to be displayed on the karaoke screen at fixed intervals side by side with the lyrics. Also add one or more document-wide annotations, such as recording information for the original, and some hypertext links to songs that are similar for vocal range, style or orchestration.

As mentioned, no straightforward and palatable XML rendition of all this information may exist. We could possibly store a few information using microformats or RDFa, as long as they can be aligned to the markup and it is a single fact per element (for instance we may have to choose between dealing with the timing information ***or*** the chords progression ***or*** the fun facts). We could adopt some of the new powerful features of overlapping syntaxes such as TexMecs' virtual elements for different sequences of the stanzas depending on the performer, or LMNL's structured attributes for fun facts annotations.

But some problems still cannot be solved with embedded markup in a purely declarative way, i.e., without recurring to annotations that must be interpreted procedurally to provide specific rendering effects. For instance, there is no easy way to have the chorus contained once and appearing several times in several positions, and no easy way to refer to the masculine or feminine lyrics depending on the gender of the singer. Even the restricted generalized order-descendant directed acyclic graphs (r-GODDAG) introduced to formally back up overlapping structures do not allow repeated or out-of-order fragments.

Finally, but not less importantly, embedded markup does not provide a simple and unambiguous way to express facts about the document as a whole, such as, for instance, the title, the authors, the year of composition, or document-level references to other documents. Each vocabulary may introduce its own syntax for such facts, possibly in attributes, or in a metadata section at the beginning of the document (as in XHTML or TEI), or in a section called "meta", or even in content-level elements that cannot be told apart of content wrappers except through a semantical analysis of their role and purpose.

3 EARMARK

The problems described in the previous section derive mostly from the very act of embedding annotations: multiple overlapping annotations, especially when referring to the same text multiple times and reordering the document order, do not naturally fit in a

[2] as accessed at the following URIs:

Beatles	http://www.youtube.com/watch?v=96YQdiMV-Jc
K. Lattimore	http://www.youtube.com/watch?v=vR96A6Hw5Nk
B. Manilow	http://www.youtube.com/watch?v=GCqRo3UmpMs
P. Carman	http://www.youtube.com/watch?v=Q1Y1j62OlXI
Scorpions	http://www.youtube.com/watch?v=LtmswOxAgKk
C. Francis	http://www.youtube.com/watch?v=9cCaLpKITyI
M. Wells	http://www.youtube.com/watch?v=ruP5E21rl54
E. Phillips	http://www.youtube.com/watch?v=axNNxMEa744
P. Bennett	http://www.youtube.com/watch?v=kTTCQ1J9GqM
Fr. Distinction	http://www.youtube.com/watch?v=lUbhDnziyi0
Patty Pravo	http://www.youtube.com/watch?v=trQwHhtkC7o

linear text structure, and analogously there is no natural position for embedding annotations to the whole document.

On the other hand, RDF annotations do not change the annotated resource in any way, but refer to it via URIs. The problem we face in this case is that there exists no URI that refers to a fragment of text that is not wrapped within an XML or XHTML element provided with an ID. And since XHTML or XML elements need to follow a nice, hierarchical, document-order-compatible structure, we are back to the beginning with the problem of overlapping hierarchies that play with multiplicities and reshuffling of the document order.

In fact a proposal for URIs to arbitrary pieces of text and XML documents exists based on the XPointer schema of the XPointer W3C standard. Grounding RDF annotations on fragments of text specified by URIs with XPointer would free the annotations from dealing with the limitations of embedded markup.

Unfortunately this schema was never standardized by the W3C and there is no sign that it will ever be in the foreseeable future. Furthermore, from the RDF point of view all URIs are opaque strings referring to different resources, and as such it would be difficult to create ontologies and make inferences that differentiate assertions on text fragments from assertions on elements or other structures, or that would allow us to verify on overlapping or superimposition of assertions.

Thus EARMARK (Extreme Annotational RDF Markup) is proposed to create assertions on text fragments by using an intermediate ontology that subsumes the XPointer schema and builds from there the concept of elements and generic identifiers.

3.1 Basic XPointer classes in EARMARK

The *Extreme Annotational RDF Markup* (*EARMARK*) is defined through an OWL document specifying the classes of the model. Through EARMARK we can produce documents containing assertions about the individuals that instantiate the defined ontology.

The classes introduced in this section create a *flat* and *externalized* characterization of the markup of one or more documents. In order to understand the conceptualization of an EARMARK document, we will be using systematically the N3 syntax [4].

An EARMARK document defines its own namespace:

```
@prefix :       <http://www.essepuntato.it/2008/12/earmark#> .
```

The textual content of a EARMARK document is conceptually separated from the annotations, and is referred to by means of assertions on the specific class called "Docuverse". The name is based on the concept used by Ted Nelson in his Xanadu Project [15] to refer to the collection of text fragments that can be interconnected to each other and transcluded in new documents.

The individuals of this class represent the object of discourse, i.e. all the text containers related to a particular EARMARK document.

```
:Docuverse
      a owl:Class ;
      rdfs:subClassOf owl:Thing .
:has-uri
      a owl:DatatypeProperty , owl:FunctionalProperty ;
      rdfs:domain :Docuverse ;
      rdfs:range xsd:anyURI .
:has-text
      a owl:DatatypeProperty , owl:FunctionalProperty ;
      rdfs:domain :Docuverse ;
      rdfs:range xsd:string .
```

Any individual of the Docuverse class – commonly called a docuverse in all lowercase to distinguish it from the class – either contains or refers to the text fragments that are the content of the document. This corresponds to one of the two defined properties, *has-uri* if the content is stored at a particular URI and *has-text* if we want to have the text as a string inside the EARMARK document.

A location is the expression of a position in a particular docuverse. In order to specify it, we introduce a class called "Location". This will be used to determine the extremes of a range, with the conceptual model introduced by the XPointer schema of XPointer [7] and specified by the property *at* to define a precise point in the docuverse that is indicated by the property *refers-to*.

```
:Location
      a owl:Class ;
      rdfs:subClassOf owl:Thing .

:refers-to
      a owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Location ;
      rdfs:range :Docuverse .

:at
      a owl:DatatypeProperty , owl:FunctionalProperty ;
      rdfs:domain :Location ;
      rdfs:range xsd:string .
```

The value for the property *at* is technically a string, in order to use any possible language to identify a particular position. For the purpose of this paper, we will refer to the XPointer model.

The expressions and the corresponding semantics is straightforward: the expression "`xpointer(point(.42))`", for instance, indicates the cursor in-between the 42nd and the 43rd character; with "`xpointer(point(/1/9.3))`" we mean the cursor between the 3rd and the 4th character of the ninth node of the root, and so on.

We then define the class Range for anything lying between two locations:

```
:Range
      a owl:Class ;
      rdfs:subClassOf owl:Thing .
:begins
      a owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Range ;
      rdfs:range :Location .
:ends
      a owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:domain :Range ;
      rdfs:range :Location .
```

A range, i.e, an individual of the class Range, is defined by a starting and an ending location through the properties *begins* and *ends* respectively. In order to define a proper range, these locations must refer to the same docuverse[3].

There is no restriction on the locations used for the *begins* and *ends* properties, and in fact a peculiar advantage of this approach is that it is possible to define ranges *according to* as well as *opposite to* the text order of the docuverse they refer. For instance, the string "desserts" can be considered both in document order, with the *begins* location lower than the *ends* location, or in opposite order, forming "stressed", an interesting example of *semordnilap*[4]. So, the locations associated to a particular range through the properties "begins" and "ends" define the way the range must be read.

[3] Note that this particular restriction is not directly expressible in OWL.

[4] http://en.wikipedia.org/wiki/Palindrome#Semordnilaps

3.2 Markup classes in EARMARK

The classes introduced in the previous section are a flat representation of an EARMARK document: since ranges are not organized hierarchically but sequentially, they can freely overlap one another.

In this section we introduce three more classes that combine the ranges and the other elements in semantically full markup annotations, providing a clear containment model and (if needed) a hierarchical structure to the various parts of the EARMARK document.

The class "MarkupItem" is the abstract superclass defining artifacts that should be interpreted as markup. The N3 implementation is the following:

```
:MarkupItem
      a owl:Class ;
      rdfs:subClassOf owl:Thing .
:has-general-identifier
      a owl:DatatypeProperty , owl:FunctionalProperty ;
      rdfs:domain :MarkupItem ;
      rdfs:range xsd:string .
```

A MarkupItem individual is a sequence, i.e. an `rdf:Seq`, of individuals belonging to the classes MarkupItem and Range. It can have a name specified by the property "has-general-identifier" (general identifier was the term used in the SGML word to refer to the name of the elements [8]).

MarkupItem is sub-classed in two disjointed classes: "Element" and "Attribute". We need these classes to obtain a more precise characterization of markup items. An element is a markup item that contains other structures and/or text, while an attribute represents, in a way, metadata about the markup item containing it.

```
:Attribute
      a owl:Class ;
      rdfs:subClassOf :MarkupItem ;
      owl:disjointWith :Element .
:Element
      a owl:Class ;
      rdfs:subClassOf :MarkupItem ;
      owl:disjointWith :Attribute .
```

Through this classification we can descibe all the concepts introduced by XML, LMNL or TexMecs, including virtual elements [17,18], structured attributes, anonymous elements [20] and so on.

4 EXTERNALIZING AND EMBEDDING MARKUP

In this section we address the problem of converting EARMARK into XML (or TexMecs or LMNL) and vice versa. Since EARMARK is more expressive than embedded markup languages, the externalization of embedded markup items into EARMARK assertions is automatic and straightforward.

On the other hand, the opposite operation of embedding EARMARK annotations is in general not possible. For this reason, one of the key aspects of embedding EARMARK is the selection of the largest subset of assertions that can actually be embedded, and the decision about the destiny of what could not.

A number of approaches can be thought of for dealing with these remaining assertions, up to its embedding in the XML document as a RDF/XML fragment [3]. These approaches constitute therefore a system of tricks to force any set of EARMARK assertions into an embedded syntax.

4.1 Externalizing markup into EARMARK

Generally speaking, the translation process from a document written in a particular markup model, such as XML, into an EARMARK document is fully automatic. Particularly, referring to XML, the algorithm that translates into EARMARK is composed of the following steps:

- The creation of one or more docuverses depending on the number of data we must handle;
- The identification of the ranges relative to the docuverses;
- The identification of the *leaf* markup items, i.e. those containing attributes and ranges only;
- The identification of the *internal* markup items, i.e. those either containing other markup items or a mixed content of markup items and ranges.

As an example, let us consider a fragment (using a simplified TEI grammar) of the lyrics of "And I Love Her" by the Beatles.

```
<text>
    <body>
        <head>And I love <span alt="him">her</span></head>
        <lg type="verse" n="1">
            <l>I give <span alt="him">her</span> all my love</l>
            <l>That's all I do</l>
            <l>And if you saw my love</l>
            <l>You'd love <span alt="him">her</span> too</l>
        </lg>
        <lg type="refrain">
            <l>I love <span alt="him">her</span></l>
        </lg>
        ...
    </body>
</text>
```

In EARMARK strings are placed in one or more docuverses. As mentioned, there are two different types of docuverses, i.e., autonomous resources (i.e., independent files identified by a URIs, which is a potentially appropriate design choice for the actual content of the document) and local strings (i.e. an internal data value, potentially appropriate for those strings without an independent existence, such as attribute values, metadata, and so on).

For the XML version of "And I Love Her", we will employ six docuverses:

- an independent text file with the lyrics. Gender-specific differences in the lyrics are expressed in the most appropriate position to keep the text file readable autonomously.

```
And I love her / him
I give xxx all my love
That's all I do
And if you saw my love
You'd love xxx too
```

- a local string containing strings for all attribute values ;
- a local string for the values of metadata elements such as title, author, creation dates, etc.;
- a local string containing times for the appearance of the individual verse lines;
- an independent file with a selection of fun facts. This could just as well be an existing HTML resource such as the one in http://www.songfacts.com/detail.php?id=43;
- a local string containing values for the chords of the song.

Note that we immediately introduce the machinery for overlapping elements and shared text fragments in the example given by the genderized versions of the lyrics. In a way, we are already deconstructing the attribute `alt` in the XML version as a

trick to force the embedding of alternative texts, something that is handled natively by EARMARK.

Note also that we can add any additional annotation (such as spaces, separators, etc.) to each docuverse content in order to make it more readable autonomously. That is possible because through EARMARK we can specify only particular ranges, explicitly ignoring some text content of the docuverses. For instance, the N3 translation of the first two docuverses is:

```
e:lyrics
    a :Docuverse ;
    :has-uri
"http://www.essepuntato.it/2009/01/andiloveher.txt"^^xsd:anyURI .
e:attribute_values
    a :Docuverse ;
    :has-text "refrain - verse - 1 - 2 - 3 - 4"^^xsd:string .
```

All the strings we use to define the real text content of an EARMARK document are identified by ranges. Ranges refer to any of the docuverse, and can overlap and invert order. For example, the ranges for the song title and the refrain overlap over the same range.

We next define a range for each text node of the XML document contained in an element or in an attribute, e.g.:

```
e:r_title_common
    a :Range ;
    :begins e:location0-lyrics ;
    :ends e:location11-lyrics .
e:r_her
    a :Range ;
    :begins e:location11-lyrics ;
    :ends e:location14-lyrics .
e:r_him
    a :Range ;
    :begins e:location17-lyrics ;
    :ends e:location20-lyrics .
e:r_attr_refrain
    a :Range ;
    :begins e:location0-attribute_values;
    :ends e:location7-attribute_values.
e:location0-lyrics
    a :Location ;
    :refers-to e:lyrics;
    :at "xpointer(point(.0))"^^xsd:string .
e:location11-lyrics
    a :Location ;
    :refers-to e:lyrics;
    :at "xpointer(point(.11))"^^xsd:string .
e:location14-lyrics
    a :Location ;
    :refers-to e:lyrics;
    :at "xpointer(point(.14))"^^xsd:string .
e:location17-lyrics
    a :Location ;
    :refers-to e:lyrics;
    :at "xpointer(point(.17))"^^xsd:string .
e:location20-lyrics
    a :Location ;
    :refers-to e:lyrics;
    :at "xpointer(point(.20))"^^xsd:string .
e:location0-attribute_values
    a :Location ;
    :refers-to e:attribute_values ;
    :at "xpointer(point(.0))"^^xsd:string .
e:location7-attribute_values
    a :Location ;
    :refers-to e:attribute_values ;
    :at "xpointer(point(.7))"^^xsd:string .
```

Some ranges can be used more than one time in the final EARMARK document. For instance, both the "r_title_common" and the "r_her" ranges are used both in the title of the song and in the lines of the refrain.

Using these ranges we can now create the leaf markup items, i.e. all the attributes or all the *first-level* elements. The latter are all the elements that have a simple content, i.e., sequences of ranges and attributes only.

Given an XML markup item *E*, an EARMARK markup item is made as follows:

- if *E* has an identifier specified through the attribute "id", then the resulting markup item has the same identifier. Otherwise, a random identifier is generated;
- the name of *E* is the general identifier;
- all attributes of *E* are translated into individual markup items. They are recursively generated with these same rules;
- the attributes and the ranges corresponding to the text content end up as the sequence of the new markup item.

For instance, the N3 translation of the attribute `type` and of the `l` elements of the refrain of both the feminine and masculine version of the song, using the ranges previously defined, are:

```
e:attr_type_refrain
    a :Attribute , [a rdf:Seq ; rdf:_1 e:r_attr_refrain] ;
    :has-general-identifier "type"^^xsd:string .
e:l_refrain_f
    a :Element , [a rdf:Seq; rdf:_1 e:r_title_common; rdf:_2
e:r_her] ;
    :has-general-identifier "l"^^xsd:string .
e:l_refrain_m
    a :Element , [a rdf:Seq ; rdf:_1 e:r_title_common ; rdf:_2
e:r_him]
    :has-general-identifier "l"^^xsd:string .
```

The difference between those leaf elements that are just sequences of ranges only and those that are sequences of attributes and ranges mirrors the difference between types in XML Schema [21], with the former resembling simple type elements with simple content, and the latter resembling complex type elements with simple content and attributes.

The process to follow to generate the inner markup items is the same as previously shown, except for the fact that the sequence does not contain ranges, but only attributes and other elements. Again, this is closely resembling complex types in XML Schema.

In figure 1 a graphical representation is shown of the line groups of the refrain in both the feminine and masculine version, while the following is the corresponding N3 rendition:

```
e:lg_refrain_f
    a :Element , [a rdf:Seq ; rdf:_1 e:attr_type_refrain ; rdf:_2
e:l_refrain_f] ;
    :has-general-identifier "lg"^^xsd:string .
e:lg_refrain_m
    a :Element , [a rdf:Seq ; rdf:_1 e:attr_type_refrain ; rdf:_2
e:l_refrain_m] ;
    :has-general-identifier "lg"^^xsd:string .
```

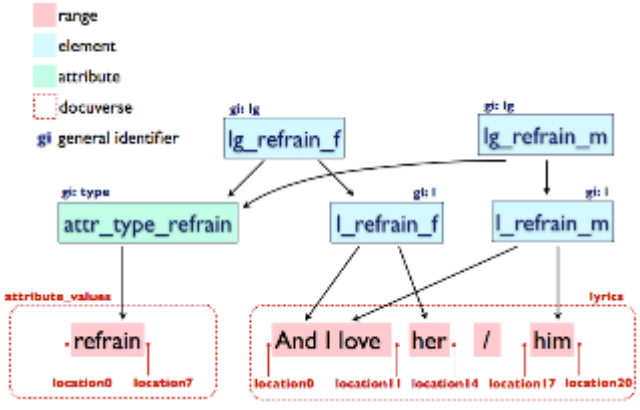

Fig. 1: A graph representation of EARMARK assertions

Obviously, EARMARK expressiveness is greater than XML's: while the structure externalized from the XML document is just a tree, through EARMARK we can express general digraphs with or

without repeatable edges depending on the particular context we are taking into consideration.

Through such digraphs we can handle particular scenarios that involve overlapping – when some ranges are overlapped between them – or virtual elements – when non-contiguous ranges are contained by a particular markup item. Some of them were already shown with the genderized version of the lyrics.

Even if some languages, such as TexMECS and LMNL, can handle these situations, there are a lot of other scenarios that cannot be addressed by any existing markup model and yet they are available through EARMARK.

One of these is *repeatability* and *ordering* of items: through EARMARK we can specify if the items in a particular markup item can be repeatable or not, and ordered or not, simply by using "rdf:Seq", the "rdf:Bag" or the "rdfs:Container" container classes.

Using this feature, we can reuse any markup item or range, e.g. the pronoun "her", in different parts of the document avoiding repetitions, as shown in figure 2.

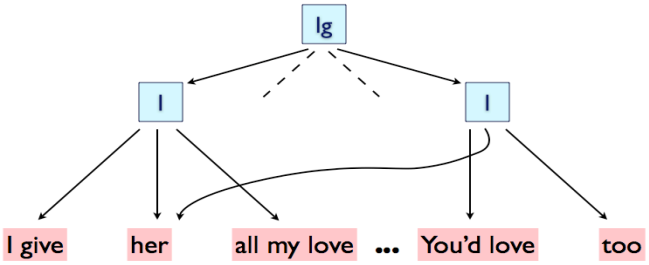

Fig. 2: reusing ranges in EARMARK

EARMARK also allows to specify mixed sequence of attributes, elements and ranges in any order. This implies that, differently from XML, LMNL and TexMECS, EARMARK can have sequence in which attributes, elements and ranges are freely mixed in any order. For example, considering A_i as attributes and E_j as elements, a markup item can be generated as an ordered sequence to contain, in order, E_1, E_2, A_1, E_3, and A_2. Moreover, the same general identifier can be specified for multiple attributes in the sequence. None of these situations is possible in any embedded markup models.

4.2 Choosing what to linearize

The opposite process, that of generating a linearized structure (such as an XML document) from a set of EARMARK annotations is less straightforward, mostly because of the substantially greater expressive power of EARMARK annotations. Without loss of generality, we will be describing a conversion to XML, since converting to LMNL or TexMecs will constitute a much simpler exercise of stopping the linearization a few steps earlier.

Although the conversion of any EARMARK subset that already describes a tree is obviously immediate and fully automatic, several different options exist for any further EARMARK annotations that we wish to linearize. Since these additional annotations are at odds with a tree-like structure, we need to use a few tricks to obtain a well-formed XML document, and of course the choice of tricks to use is wide and rich. In this section we will explore the task of linearizing a chosen tree-shaped subset of the EARMARK document, and in the following section we will describe a few options for the remaining assertions.

The construction of the tree we envision is bottom up:

- The first step is deciding which docuverses (or fragments thereof) will constitute the content of the document, which the content of the attributes, and which, if any, will be ignored.
- Then a subset of the first-level elements needs to be chosen as well as the ranges they contain. Of course, no overlapping or reverse order ranges can be accepted as such.
- There might well be the situation whereby multiple independent sets of first-level elements exist, each of which is by itself non-overlapping, but combined with others would. In this situation, of course, only one set can be selected as the main hierarchy, and all others will need to employ a trick to be expressed in the final linearized document. One possible way to do so is to create independent sets of elements and hierarchy over elements, and then choose the largest set as composing the principal hierarchy, and all others as candidates for tricks.
- Mixed content elements are sequences of ranges and first-level elements, and are generated once all contained elements are ready.
- Similarly, structure elements (only containing other elements) are available for creation once their content is already generated.
- Finally, attributes and their ranges are selected as well and converted into linearized form and associated to their elements.
- The final result of this linearization is possibly a selection of separate and disjoint trees, each linearizing a connected component of the EARMARK document. It is then an linearization choice either to generate several independent XML documents or to employ the *universal root* pattern[5] and include these structures within a single *new* element that becomes their container.

Whatever is left out of this linearization process needs to be approached using one or more of the methods described in the next section.

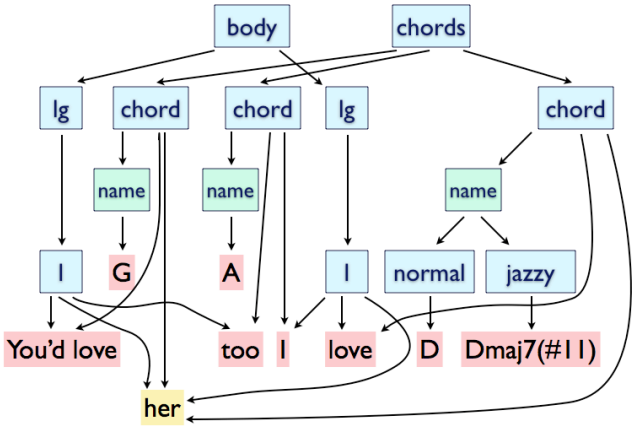

Fig. 3: a complex EARMARK graph

4.3 Options for non-linearized annotations

Some kinds of EARMARK structures are not directly linearizable by embedding. In order to allow a full representation of a complex EARMARK document such as the one shown in figure 3 we

[5] http://www.xmlpatterns.com/UniversalRootMain.shtml

therefore need to apply some stratagem to force the hierarchical structure to accept these *left out structures*.

Reasonably, frequent left out structures would include:

- overlapping leaf elements referring to contiguous ranges;
- overlapping leaf elements referring to non-contiguous ranges;
- shared ranges;
- text variants;
- overlapping inner elements;
- structured attributes.

In 4.2 we listed the EARMARK assertions that could not be directly translated into the TEI document. Let us examine a few potential approaches (which we call *trick*) for forcing the conversion. A few of such approaches, as well as algorithms for passing from one to the other, and described in [14].

4.3.1 Milestones

Plain overlapping leaf elements (i.e. elements that partially share the text content, but no lower structures) may be forced into an XML structure via *milestones* as proposed in CLIX [6].

The open and close tags of the unconverted elements are considered as individual empty elements placed in the positions where they should reside. The attribute role specifies whether the empty element corresponds to a start or end tag, and the sID and eID attributes connect the two elements in a single conceptual one.

```
<body>
    <lg>
        <l>
            <chord name="G">You'd love her</chord>
            <chord name="A" clix:role="start-range" clix:sID="A"/>
            too
        </l>
    </lg>
    <lg>
        <l>
             I
             <chord name="A" clix:role="end-range" clix:eID="A"/>
             <chord name="D">love her</chord>
        </l>
    </lg>
</body>
```

Although easy to implement and appreciate, milestones are nonetheless limited in that only frontier overlapping (i.e., overlapping on ranges) is expressible.

4.3.2 Fragmentation

Another approach is to use fragmentation as introduced by the TEI guidelines [17].

Overlapping elements are separated in many multiple fragments each of which properly nests within their container. Individual fragments are then connected via attributes such as *next* or *previous*.

```
<body>
    <lg>
        <l>
            <chord name="G">You'd love her</chord>
            <chord name="A" xml:id="a1" next="a2">too</chord>
        </l>
    </lg>
    <lg>
        <l>
             <chord name="A" xml:id="a2">I</chord>
             <chord name="D">love her</chord>
        </l>
    </lg>
</body>
```

4.3.3 Repetitions

The easiest trick for dealing with shared ranges is simply to multiply the instances of the corresponding text and possibly annotate that all instances except the first one is redundant.

```
<chord name="G">You'd love <seg repeat="r_her">her</seg></chord>
```

4.3.4 Hidden variants

When we have multiple variants of the same text, we may want to hide in substructures (such as attributes or subelements) the alternative variants. One such approach is provided by TEI [17]:

```
<chord name="G">You'd love <seg alt="him">her</seg></chord>
```

4.3.5 RDFa

RDFa [2] allows arbitrary assertions to be placed on existing elements. It is understood that if an assertion exists over a text fragment that is not wrapped within an existing element, a generic element (such as the HTML *span* or the TEI *seg*) is added to allow for RDFa assertions to attach to the corresponding content.

For instance, support for overlapping inner structures are difficult to provide in either fragmentation or milestones, but become possible in RDFa. Consider for instance the sequence which contains individual *chord* elements and overlaps with the *lg* element containing individual *l* elements.

RDFa thus supports the specification of a virtual instance of the class Chords, expressed as a sequence of three instances of the Chord class (in fact, one instance each of subclasses GChord, AChord and DChord of the Chord class) as follows:

```
<body about="#Chs" typeof="#Chords">
    <lg typeof="rdf:Seq" property="rdf:_1" href="#G">
        <l property="rdf:_2" href="#A">
            <seg about="#G" typeof="#GChord" property="#has">
                You'd love her
            </seg>
            <seg
                about="#A" typeof="#AChord" property="#has-first-
part">
                too
            </seg>
        </l>
    </lg>
    <lg property="rdf:_3" href="#D">
        <l>
             <seg about="#A" property="#has-second-part">I</seg>
             <seg about="#D" typeof="#DChord" property="has">
                 love her
             </seg>
        </l>
    </lg>
</body>
```

4.3.6 Embedded RDF

When all else fails, the fallback approach is simply to place the remaining assertions as an RDF/XML block in the XML structure, either in a block properly thought out for external vocabularies, or converted into some local vocabulary, or even as a lump of XML elements placed in a random position within the document.

This is useful, for instance, for dealing with structured attributes *a la* LMNL [20]. In the following example, a RDF block is inserted in the TEI document to provide support for the attribute *name* of the *chord* element, which contains a structure of two different values wrapped by elements *normal* and *jazzy*. This allows the *name* of the *chord* to cater for both a pop and a jazz rendering of the tune, while remaining one attribute of one element.

```
<body>
  <rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.essepuntato.it/2008/12/earmark#">
       <Attribute rdf:about="#a_name_structured">
          <has-general-identifier rdf:datatype="xsd:string">
            name
```

```
        </has-general-identifier>
        <rdf:type>
          <rdf:Seq>
              <rdf:li rdf:resource="#normal"/>
              <rdf:li rdf:resource="#jazzy"/>
          </rdf:Seq>
        </rdf:type>
      </Attribute>
      <Element rdf:about="#normal">
        <has-general-identifier rdf:datatype="xsd:string">
          normal
        </has-general-identifier>
        <rdf:type>
          <rdfs:Container>
              <rdf:li rdf:resource="#r_D"/>
          </rdfs:Container>
        </rdf:type>
      </Element>
      <Element rdf:about="#jazzy">
        <has-general-identifier rdf:datatype="xsd:string">
          jazzy
        </has-general-identifier>
        <rdf:type>
          <rdfs:Container>
            <rdf:li rdf:resource="#r_Dmaj7"/>
          </rdfs:Container>
        <rdf:type>
      </Element>
  <rdf:RDF>
  ...
</body>
```

5 CONVERTING DOCUMENTS

The transformation processes introduced in Section 4 have been implemented through two (rather complex) XLST documents. We have realized a tool that provides a full translation into and from an EARMARK document. By "full translation" we mean an non-entropic and complete transformation that allows *roundtrips*, i.e., that converting a converted document returns a document that is identical (or at least very similar) to the original one. Because of the different expressiveness between those languages, the efficacy and complexity of the roundtrip conversions are quite different.

Obviously, the roundtrip XML → EARMARK → XML is easy to realize, since XML is less powerful than EARMARK: the EARMARK document obtained off the XML one contains nothing more and nothing less than the full assertions of the original XML document, that pose no problems in translating them back into XML elements and attributes, without the need for any trick.

On the contrary, the EARMARK →XML → EARMARK process is much more complex. Fitting all the non-XML-expressible data of an EARMARK document into XML one requires the choice of any of a number of tricks as discussed in section 4.3. These tricks need to be documented in the converted document so as to be able to perform the correct opposite transformation.

Figure 4 shows the main interface of a web-based tool to perform the conversions online. The key issue of the interface is to allow the user to choose the tricks to use for representing non-XML-expressible assertions.

The tool allows users to choose one of the possible roots by specifying the associated namespace. That root will represent the document element of the resulting XML document, and the principal XML vocabulary. Overlapping structures will be rendered using any of the tricks.

Starting from the ranges, the tool recursively linearizes its container elements and attributes, maintaining a tree structure. The character encoding of the external docuverses used in the linearization is established by relying on existing annotations or on the usual heuristic approaches, but our implementation always generates documents using UTF-8 strings. After reaching the text ranges, any remaining EARMARK assertion will be embedded using the appropriate trick, chosen among the allowed ones.

Since the user can select an incomplete subset of all the possible

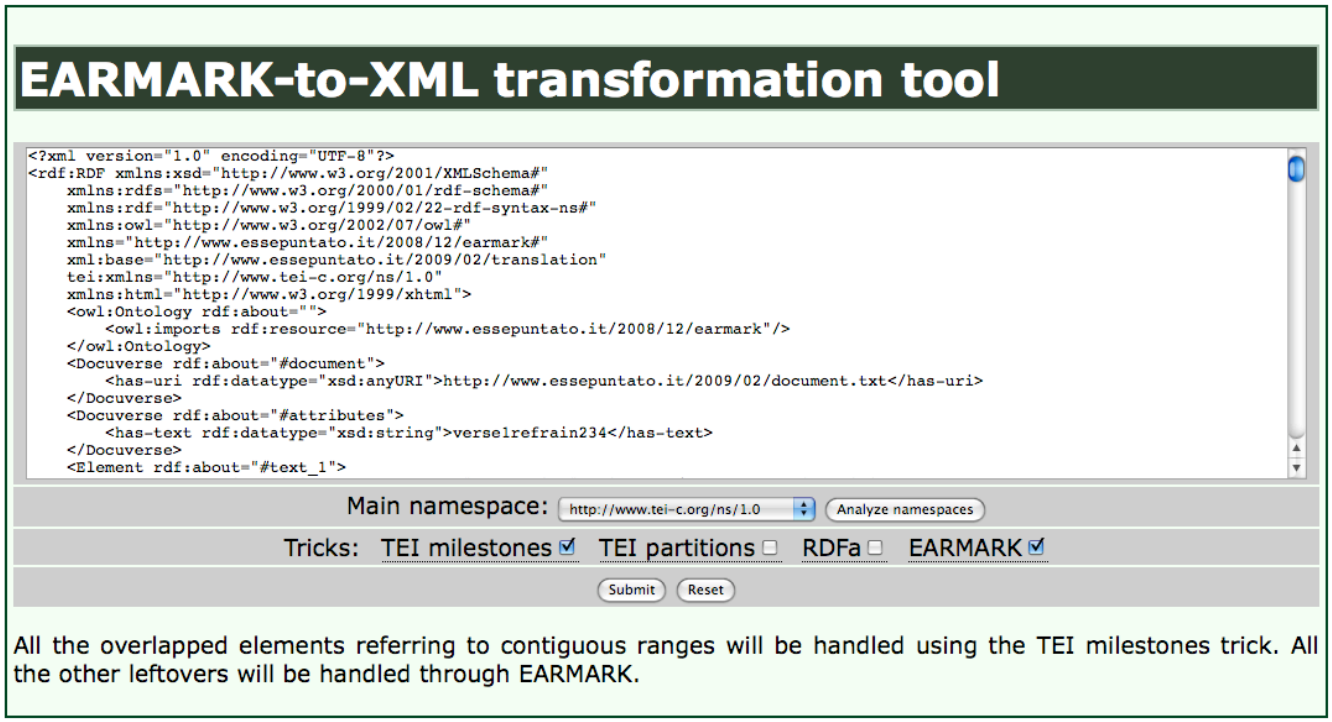

fig. 4: The main interface of the web-based conversion tool between EARMARK and XML

tricks, it is not guaranteed that all the leftovers can be put in the resulting XML document with just the selected tricks. For example, if we choose TEI milestones and RDFa, we can express the overlapping elements for contiguous and non-contiguous ranges, but structured attributes are not expressible.

In such case, the ability to observe the non-entropic constraint strongly depends on the complexity of the starting document and on the set of tricks that the user selected.

6 CONCLUSIONS

In this paper we presented the Extreme Annotational RDF Markup (EARMARK), a proposal for integrating the advantages of external annotations and embedded markup into a single unifying framework.

Through EARMARK authors can express a large number of markup assertions and observations that would otherwise be non expressible, including overlapping elements, elements over non-contiguous ranges, structured attributes, etc.

In this paper we concentrated on making sure that EARMARK and traditional markup language can coexist and talk to each other, and that a conversion path exists even for particularly convoluted EARMARK expressions.

In further work we plan both to increase the number and sophistication of the EARMARK tools, and to explore, both formally and pragmatically, the expressive power of the EARMARK language and its applications.

7 ACKNOWLEDGEMENTS

The authors would like to thank the participants to the Workshop on Multi-Dimensional Markup held in Amsterdam in December 2008, and in particular Claus Huitfeld, Michael Sperberg-McQueen and Jeni Tennison for the fruitful discussions. The authors also recognize endless credit to Angelo Di Iorio, Paolo Marinelli, and Stefano Zacchiroli for providing the solid foundations to this work.

8 REFERENCES

[1] Adida, B. & Birberck, M. (2008). RDFa Primer, Bridging the Human and Data Webs. W3C Working Group Note. W3C Recommendation. World Wide Web Consortium. “http://www.w3.org/TR/xhtml-rdfa-primer/”.

[2] Adida, B., Birbeck, M., McCarron, S., & Pemberton, S. (2008). RDFa in XHTML: Syntax and processing. W3C Recommendation. World Wide Web Consortium. “http://www.w3.org/TR/rdfa-syntax/”.

[3] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., & Stein, L. A. (2004). OWL Web Ontology Language Reference. W3C Recommendation. “http://www.w3.org/TR/owl-ref/”.

[4] Berners-Lee, T. (2004). Notation 3, a readable RDF syntax. “http://www.w3.org/DesignIssues/Notation3”.

[5] Burnard, L., Sperberg-McQueen, M., & Bauman, S. (2001). A Gentle Introduction to XML. Burnard and Sperberg-McQueen, eds. Guidelines for Electronic Text Encoding and Interchange. TEI Consortium. “”http://www.tei-c.org/Guidelines/P4/html/SG.html.

[6] DeRose. S. (2004) Markup overlap: A review and a horse. In the Proceedings of Extreme Markup Languages 2004. Montreal, Canada.

[7] DeRose, S., Maler, E., & Daniel, R. (2002). XPointer xpointer() Scheme. W3C Working Draft. World Wide Web Consortium. “http://www.w3.org/TR/xptr-xpointer/”.

[8] Goldfarb, C. F. (1990). The SGML Handbook. Oxford University Press, USA.

[9] Grosso P., Maler M., Marsh J., Walsh N., XPointer Framework. W3C Recommendation. World Wide Web Consortium. “http://www.w3.org/TR/xptr-framework/”

[10] Klyne, G., Carroll, J. J., & McBride, B. (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation. World Wide Web Consortium. “http://www.w3.org/TR/rdf-concepts/”.

[11] Huitfeldt, C. & Sperberg-McQueen, C. M. (2001). TexMECS: An experimental markup meta-language for complex documents. Working paper of the project Markup Languages for Complex Documents (MLCD), University of Bergen.

[12] Manola, F. & Miller, E. (2004). RDF Primer. W3C Recommendation. World Wide Web Consortium. “http://www.w3.org/TR/REC-rdf-syntax/”.

[13] Marcoux, Y. (2008). Graph characterization of overlap-only TexMECS and other overlapping markup formalisms. In the Proceedings of Balisage: The Markup Conference 2008.

[14] Marinelli, P., Vitali, F., & Zacchiroli, S. (2008). Towards the unification of formats for overlapping markup. New Review of Hypermedia and Multimedia, Volume 14, Issue 1 January 2008 , pages 57–94.

[15] Nelson, T. (1980). Literary Machines: The report on, and of, Project Xanadu concerning word processing, electronic publishing, hypertext, thinkertoys, tomorrow's intellectual... including knowledge, education and freedom. Mindful Press, Sausalito, CA, USA.

[16] Nelson, T. (1997) Embedded markup considered harmful. World Wide Web Journal, 2(4):129–134.

[17] Sperberg-McQueen, C. M. & Burnard, L. (2005). TEI P5 Guidelines for Electronic Text Encoding and Interchange (revised). The Association for Computers and the Humanities. “http://www.tei-c.org/Guidelines/P5/”.

[18] Sperberg-McQueen, C. M. & Huitfeldt, C. (2004). GODDAG: A Data Structure for Overlapping Hierarchies. In the Proceedings of the 8th International Conference on Digital Documents and Electronic Publishing, DDEP 2000 Munich, Germany.

[19] Sperberg-McQueen, C. M. & Huitfeldt, C. (2008). Markup Discontinued: Discontinuity in TexMecs, Goddag structures, and rabbit/duck grammars. In the Proceedings of Balisage: The Markup Conference 2008. Montreal, Canada.

[20] Tennison, J. & Piez, W. (2002). The Layered Markup and Annotation Language (LMNL). Paper presented at the Late breaking paper presented at Extreme Markup. Montreal, Canada.

[21] Thompson, H. S., Beech, D., Maloney, M., & Mendelsohn, N. (2001). XML Schema Part 1: Structures. W3C Recommendation. World Wide Web Consortium. “http://www.w3.org/TR/xmlschema-1/”