Annotations with EARMARK in practice: a fairy tale

Gioele Barabucci
Department of Computer Science and Engineering
University of Bologna
barabucc@cs.unibo.it

Angelo Di Iorio
Department of Computer Science and Engineering
University of Bologna
diiorio@cs.unibo.it

Silvio Peroni
Department of Computer Science and Engineering
University of Bologna
essepuntato@cs.unibo.it

Francesco Poggi
Department of Computer Science and Engineering
University of Bologna
fpoggi@cs.unibo.it

Fabio Vitali
Department of Computer Science and Engineering
University of Bologna
fabio@cs.unibo.it

ABSTRACT

There is still a gap between models for external annotations of markup documents and their applications. In this paper we present the EARMARK API, a Java framework that allows users to combine embedded markup with stand-off markup. We discuss a few relevant issues on adding annotations to TEI documents that refer to external entities and we show how to use EARMARK to link textual ranges of such documents to resources in the Linked Open Data.

Categories and Subject Descriptors

I.7.1 [**Document and text processing**]: Document and text editingDocument management

Keywords

annotations; linked open data; stand-off markup; embedded markup; EARMARK.

1. ONCE UPON A TIME A LITTLE ANNOTATION...

Literary texts are not isolated conceptual units: they live in a complex pattern of references to people, events, locations, concepts, and of course to other documents. And yet again these references could then be connected to their justifications and to the people that originated, added or were addressed by them. The digitalization of these texts opens new potentialities for expression, and yet multiple annotations need to be allowed by multiple encoders, and users must be allowed even to annotate documents for which they do not have write permissions. It is then necessary to create flexible mechanisms that support the process of creating and sharing multiple, independent, overlapping semantic annotations on these documents.

DH-CASE '13, September 10 2013, Florence, Italy

ACM 978-1-4503-2199-0/13/09 ...$15.00.
http://dx.doi.org/10.1145/2517978.2517990 .

We find that there are two main issues to this.On the one hand, we need a way to write and store overlapping annotations, i.e. annotations that cannot be expressed within the strict hierarchical organization of tree-based documents such as XML documents. Overlapping annotations allow users to express statements that coexist on the same document, on the same content or on fragments of content that has been annotated by others, even using vocabularies and schemas that are incompatible with the original language/format of the document. Embedded and stand-off markup techniques have been widely studied in the literature and successfully deployed towards this goal [1].

On the other hand, we need a way to refer to external entities, conceptualizations that live outside of a document and that are independent of it. For instance, the *Friend Of A Friend* (*FOAF*) ontology [3]describes people and their relations to other people.Through it it is possible to associate the role of annotator of a document to a specific person, in an unambiguous way that can be further processed by software agents. Also of importance is the ability to state that different text fragments in the same document, or in a collection of documents, refer to the same univocally identified entity.

Giving users the ability to link pieces of texts to their representation elsewhere (e.g. on the Web) opens interesting possibilities for data integration, automatic reasoning and semantic analysis in the Linked Open Data [2].This aspect is even more interesting considering the content of literary texts. These texts are full of references to people, fictional characters, places, events, etc. Each of these entities might have, for instance, a description in DBPedia [15] (a community effort to extract structured information from Wikipedia and make it machine-readable through Semantic Web technologies).

In this paper we present the EARMARK API, a Java framework that allows users to mix embedded and stand-off markup approaches when annotating literary texts, and provides native support for overlapping annotations. Annotated text fragments, entities and annotators are identified by URIs in EARMARK and can be interlinked in a very flexible and expressive way. In fact, EARMARK is based on existing Semantic Web technologies, so it enables a straight and full integration of annotations within the Linked Open Data. The paper focuses on the practical aspects of EAR-

MARK and discusses how to exploit the Java API through a running example, interleaving theoretical problems, practical issues and their solutions using the EARMARK API. Further details about the overall research, the full ontological model, and some applications can be found in other, more theoretical papers about EARMARK [9, 10].

The paper is structured as follows. In section 2 we discuss the main issues and solutions in annotating literary texts and in particular TEI documents. We focus on EARMARK and explain how to use it in that context. Section 3 goes into details of the API. Section 4 introduces some more tricky issues, such as handling changes and supporting both authors' and annotators' annotations at the same time, and explain how to use EARMARK to address this problem. Before concluding, we discuss in section 5 how to convert EARMARK documents back to XML, so to be able to convert back and forth from documents stored using current formats.

2. ANNOTATION MEETS EARMARK

There are different ways to add annotations to TEI documents, each of which may be used according to particular characteristics of the document. In addition, the task of adding annotations to documents can be even more complex when multiple hierarchies need to be defined onto the same content through *overlapping tricks* [8, 16] such as milestones, twin documents, stand-off markup, etc. In order to explain these issues and the possible approaches to annotation documents, let us consider the excerpt in figure 1 from the play *The Tempest* by William Shakespeare, distributed in TEI by the Oxford Text Archive[1].

```
<body>
  ...
  <sp>
    <speaker rend="italic">Ari.</speaker>
    <ab>
      All haile, great Master, graue Sir, haile: I
          come<lb n="301"/>
      To answer thy best pleasure; be't to fly,
      <lb n="302"/>
      To swim, to diue into the fire: to ride
      <lb n="303"/>
      On the curld clowds: to thy strong bidding,
          taske<lb n="304"/>
      <hi rend="italic">Ariel,</hi> and all his
          Qualitie.<lb n="305"/>
    </ab>
  </sp>
  <sp>
    <speaker rend="italic">Pro.</speaker>
    <ab>
      Hast thou, Spirit,<lb n="306"/>
      Performd to point, the Tempest that I
      <seg type="homograph">bad</seg> thee.
      <lb n="307"/>
    </ab>
  </sp>
  ...
</body>
```

Figure 1: An excerpt of *The Tempest* by William Shakespeare encoded in XML using TEI

The excerpt in figure 1 describes two different hierarchies. On the one hand, the element *body* contains two different speeches (elements *sp*), each having its own speaker (element *speaker*) and some content (element *ab*, *anonymous block*). On the other hand, the element *body* also contains the stanzas and the lines the text is organzied in, using the elements *lb* (i.e. *line break*), which are used to delimit the extremes of the lines rather than wrap their content. A graphic representation of these hierarchies is shown in figure 2.

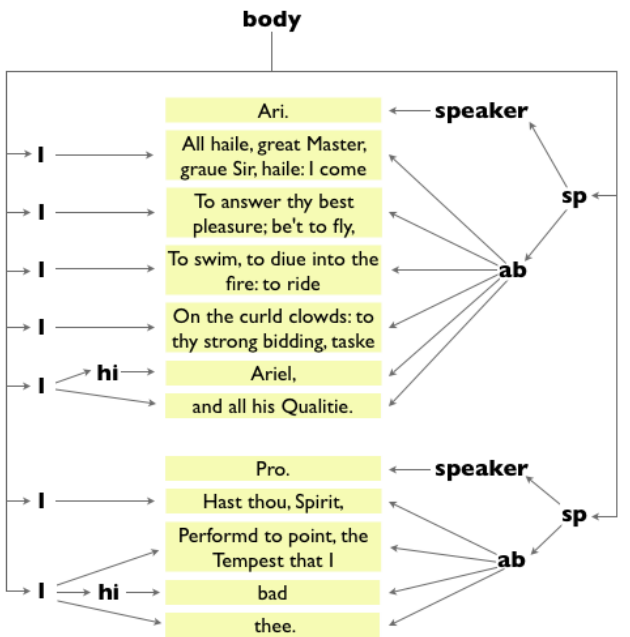

Figure 2: Two alternative TEI markup hierarchies built upon the same text content.

In the following sections we introduce the techniques currently used to add annotations to texts such as this one. After this review we introduce our EARMARK-based solution to common annotation problems.

2.1 Adding annotations

There are two ways to add annotations to existing documents: either one places the annotation within the document itself (*embedding*) or the annotations are placed in a separate document with references to the parts of the document they refer to (*standoff*). Neither embedding nor standoff annotations are inherently wrong or correct on their own; rather, each technique has its own pros and cons that must be evaluated before choosing.

In general, techniques in which the annotations are embedded have the advantage of keeping all the information in a single coherent structure within a single file. On the other hand, as the number of annotations grows, it is more and more likely for these annotations to overlap, and with them would overlap the markup used to express them. Overlapping markup is a complex topic without an universally accepted solution [1]. Another issue with embedding techniques is that they all require the permissions for the annotator to modify the original document (copying the document and then adding annotations is considered a form of standoff annotation). It is often not possible to modify the document (e.g., when it is served via read-only HTTP) or not desirable

[1]The TEI source of this play is available at `http://ota.ox.ac.uk/text/5725.xml`. The modified file is available at `http://www.essepuntato.it/2013/dhcase/thetempest.xml`.

(e.g., the author may want to publish a document but not carry the burden of publishing someone else's annotations as well).

Standoff techniques solve or greatly eases the problems of embedded annotations but raise other concerns. The main problems are all related to the need to address the data from the source original document; annotations must have a way to identify with precision the content they are referring to. This may be easy in some cases, for example where each element is explicitly addressable (e.g., RDF documents with no blank nodes) or wherethe concept of position or coordinates makes sense (e.g., in images), and complex in others (e.g., referring to a unmarked span of text within a long paragraph). Serious issues arise when the document is modified: depending on the kind of reference used, the annotations may end up out-of-sync and point to the wrong part of the document. For example, using the text in figure 1, if an annotation referred to "the line between the element `lb[@n='306']` and `lb[@n='307']`", this annotation will no longer point to the correct line when then document is changed to use speech-relative line numbers such as `n='1'` and `n='2'`.

Many techniques have been devised in the past, both for embedding annotations and standoff annotations. Usually technical solutions address only the problem of how to store the annotations, without dealing with the meaning of the annotations. In the case of embedded annotations, these solutions makes it possible to deal with overlapping markup (e.g TexMECS [12]) or offer a generic way to augment existing markup with annotations (e.g. RDFa [17]); in the case of standoff annotations, the existing technical solutions provide a way to address content (e.g. XPointer or NIF [11]).

The technical solutions are only half of what is needed to annotate documents. The other half is the use of an annotation model and vocabulary that define the meaning of each annotation.Many such vocabularies are available, ranging from very generic annotation frameworks, e.g., the Open Annotation Data Model (OADM) [22] or the older Annotation Ontology [5], to more specific frameworks, e.g., the Linguistic Annotation Framework (LAF) [13] (used to annotate the various linguistic features of a speech in its transcript) or Domeo [4] (that describes annotations used to connect scholarly documents).

2.2 Document annotations with EARMARK

The *Extremely Annotational RDF Markup*, or *EARMARK* [9, 19], is an OWL 2 DL ontology that defines document meta-markup. It is an ontologically precise definition of markup that instantiates the markup of a text document as an independent OWL document outside of the text strings it annotates, and through appropriate OWL and SWRL characterizations it can define structures such as trees or graphs and can be used to generate validity constraints (including co-constraints currently unavailable in most validation languages) [10]. In addition, using EARMARK with the *Linguistic Act* module of the *Linguistic Meta-Model* [21], it becomes possible to express and assess facts, constraints and rules about the markup structure as well as about the inherent semantics of the markup elements themselves, and about the semantics of the content of the document [20].

Using EARMARK it is possible to add annotations to the text of the TEI excerpt presented in figure 2 and then to associate fragments of that text to entities defined externally, such as those existing in the Linked Open Data. For instance, let us suppose we want to say that the strings "Ari.", "Ariel" and "Spirit" actually denote to the same fictional character, i.e. *Ariel* and its DBPedia resource `http://dbpedia.org/resource/Ariel_(The_Tempest)` . The following excerpt shows how to add these annotations using EARMARK. [2] [3]

```
# The EARMARK Document described as an OWL ontology
<http://www.essepuntato.it/example> a owl:Ontology .

# The textual content of the document to annotate
:content a earmark:URIDocuverse ;
  earmark:hasContent
    "http://ota.ox.ac.uk/text/5725.xml"^^xsd:anyURI .

# The string "Ari."
:ari-string a earmark:PointerRange ;
  earmark:refersTo :content ;
  earmark:begins "33974"^^xsd:nonNegativeInteger ;
  earmark:ends "33978"^^xsd:nonNegativeInteger .

# The string "Ariel"
:ariel-string a earmark:PointerRange ;
  earmark:refersTo :content ;
  earmark:begins "34278"^^xsd:nonNegativeInteger ;
  earmark:ends "34283"^^xsd:nonNegativeInteger .

# The string "Spirit"
:spirit-string a earmark:PointerRange ;
  earmark:refersTo :content ;
  earmark:begins "34463"^^xsd:nonNegativeInteger ;
  earmark:ends "34469"^^xsd:nonNegativeInteger .

# The strings are annotated with the same entity
:ari-string la:denotes dbp:Ariel_(The_Tempest) .
:ariel-string la:denotes dbp:Ariel_(The_Tempest) .
:spirit-string la:denotes dbp:Ariel_(The_Tempest) .
```

In the above excerpt, we use an instance of the class *earmark:URIDocuverse* to define the whole textual content of the document to annotate – in this case the Oxford Text Archive TEI version of the play *The Tempest*, available at a given URL. Once we have the content, we can thus define textual ranges upon it by defining instances of the class *earmark:PointerRange*. In EARMARK, such a range defines a textual interval by counting characters. In this case, the value of the properties *earmark:begins* and *earmark:ends* must be non-negative integers identifying unambiguous positions in the character stream.

In principle, annotations can be made by different annotators, even though in the above excerpt there is no such situation. However, in annotations environments it is important to keep track of provenance information. For instance, let us consider again the previous excerpt and let us suppose that the first *la:denotes* statement has been created by the annotator named *Silvio Peroni*, while the second and the third *la:denotes* statements have been added by another annotator, *Gioele Barabucci*. To handle such scenario, EARMARK can be used in combination with two more ontologies: the *Friend Of A Friend* (*FOAF*) ontology [3], to describe the people acting as annotators, and the *Provenance Ontology* (*PROV-O*) [14] to associate provenance information to assertions represented as instances of the class *la:LinguisticAct*.[4] The following excerpt shows how to ex-

[2] The examples uses the Turtle format.

[3] The prefixes are available at `http://www.essepuntato.it/2013/dhcase/prefixes.ttl`.

[4] The class *la:LinguisticAct* is part of the *LA-EARMARK* ontology, available at `http://www.essepuntato.it/2013/06/la-earmark`, which aligns the EARMARK and the Linguistic Act ontologies.

press the aforementioned situation:

```
:silvio a foaf:Person, prov:Agent ;
  foaf:name "Silvio Peroni" .

# Linguistic act for Silvio's annotation
:annot-by-silvio a la:LinguisticAct, prov:Entity ;
  la:hasInformationEntity :ari-string ;
  la:hasReference dbp:Ariel_(The_Tempest) ;
  prov:wasAttributedTo :silvio ;
  prov:generatedAtTime "2013-06-17T13:35:23Z"^^xsd:
       dateTime .

:gioele a foaf:Person, prov:Agent ;
  foaf:name "Gioele Barabucci" .

# Linguistic act for Silvio's annotation
:annot-1-by-gioele a la:LinguisticAct, prov:Entity ;
  la:hasInformationEntity :ariel-string ;
  la:hasReference dbp:Ariel_(The_Tempest) ;
  prov:wasAttributedTo :gioele ;
  prov:generatedAtTime
    "2013-06-18T07:31:28Z"^^xsd:dateTime .

:annot-2-by-gioele a la:LinguisticAct, prov:Entity ;
  la:hasInformationEntity :spirit-string ;
  la:hasReference dbp:Ariel_(The_Tempest) ;
  prov:wasAttributedTo :gioele ;
  prov:generatedAtTime
    "2013-06-18T07:31:28Z"^^xsd:dateTime .
```

As briefly summarized in figure 3, the *linguistic acts* are particular communicative situations including *information entities* (i.e. symbols such as strings, each conveying a meaning or denoting one or more references), *meanings* (i.e. (meta-level) objects that explain something, or are intended by something, such as linguistic definitions, logical concepts or relations, etc.) and references (i.e. individuals, sets of individuals, or facts from the world we are describing). In particular, in the example above, the information entity of each linguistic act is an EARMARK range while its reference is the DBPedia result for *Ariel*. In addition to linguistic acts, we also added data related to the agent (i.e. *prov:Agent*), in this case the particular person (i.e. *foaf:Person*), who made the linguistic act (i.e. *prov:wasAttributedTo*) and the time when it was made (i.e. *prov:generatedAtTime*).

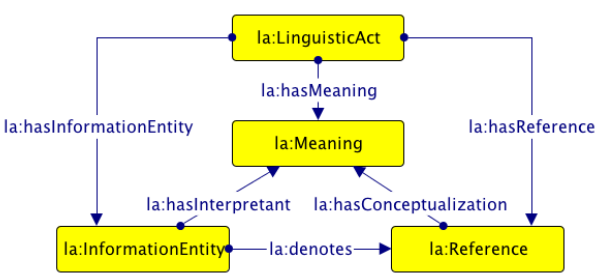

Figure 3: A Graffoo diagram of an excerpt of the *Linguistic Act* module included in the *LA-EARMARK* ontology.

The LA-EARMARK ontology allows also to specify different meanings for the same string. For instance, let us consider a new EARMARK range for the string "Master", a word pronounced by Ariel and clearly denoting another character of *The Tempest*, i.e. *Prospero*. Now, let us suppose that the two annotators we introduced before want to annotate such a range with a slightly different meaning. On the one hand, the first annotator (i.e. Silvio Peroni) wants to connote Prospero as a person; in fact Prospero is a person according to the story. On the other hand, the second annotator (i.e. Gioele Barabucci) wants to refer to Prospero as a character of the play. The following excerpts show how to model this scenario with LA-EARMARK:

```
# The string "Master"
:master-string a earmark:PointerRange ;
  earmark:refersTo :content ;
  earmark:begins "34023"^^xsd:nonNegativeInteger ;
  earmark:ends "34029"^^xsd:nonNegativeInteger .

# Silvio's interpretation of the string "Master"
:prospero-as-person a la:LinguisticAct, prov:Entity ;
  la:hasInformationEntity :master-string ;
  la:hasReference dbp:Prospero ;
  la:hasMeaning foaf:Person ;
  prov:wasAttributedTo :silvio ;
  prov:generatedAtTime
    "2013-06-18T17:23:23Z"^^xsd:dateTime .

# Gioele's interpretation of the string "Master"
:prospero-character a la:LinguisticAct, prov:Entity ;
  la:hasInformationEntity :master-string ;
  la:hasReference dbp:Prospero ;
  la:hasMeaning yago:ShakespeareanCharacter ;
  prov:wasAttributedTo :gioele ;
  prov:generatedAtTime
    "2013-06-18T17:23:23Z"^^xsd:dateTime .
```

3. DO IT IN PRACTICE: EARMARK API

We have already designed and implemented a framework for the creation, validation and manipulation of EARMARK data structures. The API is hosted on GitHub[5] under the GNU General Public License version 2.

All the code is written in Java and uses Jena.[6] The implementation of the data structure follows exactly what is defined in the EARMARK ontology, encoding OWL properties as methods of these classes.

The Java classes *EARMARKDocument*, *Range* and *MarkupItem* have been written taking into consideration a particular interface, called *EARMARKNode*, directly derived from the *Node* interface of the Java DOM implementation. This choice has been made to maintain the EARMARK data structure as close as possible to a well-known and used model for XML documents.

The API makes it simple to create/load/store/modify and add assertions to EARMARK documents directly in a Java framework. Let us take again into consideration the example introduced in section 2: in the followings excerpts, we illustrate how to add the annotations presented in section 2.2 using the API. Let us start creating a new EARMARK document and a docuverse, containing all the text content of our document[7]:

```
String ex = http://www.essepuntato.it/example/;
EARMARKDocument ed = new EARMARKDocument(
  URI.create(ex));
Docuverse doc = ed.createURIDocuverse("content",
  URI.create("http://ota.ox.ac.uk/text/5725.xml"));
```

The next excerpt shows how to create the three ranges of the example starting from the above docuverse:

```
Range ari_string = ed.createPointerRange(
  "ari-string", doc, 33974, 33978);
```

[5] https://github.com/essepuntato/EarmarkDataStructure

[6] http://jena.sourceforge.net

[7] All the excerpts of Java code introduced are downloadable at http://fpoggi.web.cs.unibo.it/DH-CASE/.

```
Range ariel_string = ed.createPointerRange(
  "ariel-string", doc, 34278, 34283);
Range spirit_string = ed.createPointerRange(
  "spirit-string", doc, 34463, 34469);
```

The next step consists in adding three annotations that specify that the ranges previously defined denote the same fictional character:

```
String la = ... /* the same as Turtle prefix */
String dbp = ... /* the same as Turtle prefix */
Model model = ed.getModel();
Property denotes = model.createProperty(
  la + "denotes");
Resource dbp_ariel = model.createResource(
  dbp + "Ariel_(The_Tempest)");
ari_string.assertsAsSubject(denotes, dbp_ariel);
ariel_string.assertsAsSubject(denotes, dbp_ariel);
spirit_string.assertsAsSubject(denotes, dbp_ariel);
```

In addition, we might want to add information about the provenance of our annotations. The following snippet shows how to add information about the name of an annotator using the EARMARK API:

```
String prov = ... /* the same as Turtle prefix */
Resource prov_Agent = model.getResource(
  prov + "Agent");
Resource silvio = model.createResource(
  ex + "silvio");
silvio.addProperty(RDF.type, FOAF.Person);
silvio.addProperty(RDF.type, prov_Agent);
silvio.addProperty(FOAF.name, "Silvio Peroni");
Resource gioele = ...
```

The next step consists in adding the information about a linguistic act: instead of adding the necessary assertions one by one, the EARMARK API allows to perform this operation in a single step by using the method *addLinguisticAct*. This method can be invoked on each *EARMARKItem* (i.e. either *MarkupItem*, *Range* or *Docuverse*), and takes as input a resource that is a *Reference*, a resource that is a *Meaning* and a resource identifying an *Agent*, and creates a linguistic act at the current time instant. In the following excerpt we show how to create the three linguistic acts introduced in section 2.2:

```
ari_string.addLinguisticAct(dbp_ariel,null,silvio);
ariel_string.addLinguisticAct(dbp_ariel,null,gioele);
spirit_string.addLinguisticAct(dbp_ariel,
                               null,gioele);
```

Finally, the scenario of last example of the previous section in which we specify different meanings for the same string can be realized in the following way:

```
String yago = ... /* the same as Turtle prefix */
Range master_string = ed.createPointerRange(
  "master-string", doc, 34023, 34029);
Resource dbp_prospero = model.createResource(
  dbp + "Prospero");
Resource character = model.createResource(
  yago + "ShakespeareanCharacters");
master_string.addLinguisticAct(
  dbp_prospero, FOAF.Person, silvio);
master_string.addLinguisticAct(
  dbp_prospero, character, gioele);
```

In this section we introduced issues and methodologies to add annotations to documents. In particular, we showed how to use EARMARK and its API to add such annotations together with provenance data about the agent that added the annotation, the *annotator*, and the time of its creation. In the next section we will deal with more more sophisticated issues and their relative solutions.

4. OTHER VILLAINS

Describing the annotation itself is only one of the problems in annotating electronic documents. Another complex issue is how to make sure that annotations still refer to the correct piece of document once the document is changed. Yet another problem is the fact that sometimes the authors can be annotators themselves, allowing the use of different techniques that may reduce the ability of others to annotate the same documents. In this section we address these additional problems with the use of various features made available by the EARMARK API.

4.1 In real life, documents do change (regardless of what philosophers think)

All the examples shown in section 2.2 implicitly assumed that the source document, i.e. the TEI version of Shakespeare's *The Tempest*, is an immutable object that does not change in the future. However this is not always the case, since the author of such a document can decide to modify something, e.g. adding some markup elements and changing some text. For instance, let us suppose that the author of our the XML in figure 1 adds the lines (elements *l*) within the element *ab* and spell out all the abbreviations in the elements *speaker* with the entire name of the character, as shown in the following excerpt[8]:

```
<sp>
  <speaker rend="italic">Ariel</speaker>
  <ab>
    <l>All haile, great Master, graue Sir, haile: I
       come</l>
    <l>To answer thy best pleasure; be't to fly,</l>
    <l>To swim, to diue into the fire: to ride</l>
    <l>On the curld clowds: to thy strong bidding,
       taske</l>
    <l><hi rend="italic">Ariel,</hi> and all his
       Qualitie.</l>
  </ab>
</sp>
```

Taking into consideration the above excerpt, what happens to the EARMARK ranges we previously made? What about the annotations defined upon them? In fact, the EARMARK pointer ranges now refer to wrong positions within the document. For instance, according to the modified document, the four ranges defined previously – i.e. *:ari-string*, *:ariel-string*, *:spirit-string* (not included in the above excerpt) and *:prospero-string* – now points to four different strings – i.e. "Arie", "his Q", "point," and "eat Ma". This is due to the absolute position EARMARK uses to create pointer ranges: just one more characters can change all the pointer ranges defined on that document.

Although EARMARK does not provide a single comprehensive solution this problem, it offers a mechanism to avoid some of the issues related to document modifications through the use of *XPath pointer ranges*. In contrast to plain pointer ranges that use the entire docuverse as the context for their offsets, an XPath pointer range allows us to specify the particular node of the whole docuverse to use as the context relative to which the offsets are expressed. Note that, by using these ranges, we implicitly admit that our docuverse is an XML structure. Moreover, the properties *begins* and *ends* must be applied on the string value obtained by juxtaposing all the descendant text nodes of the node retrived by

[8]The modified document is available at `http://www.essepuntato.it/2013/dhcase/thetempest-modified.xml`.

the XPath. The following excerpt shows how to create xpath pointer ranges so as to have correct ranges for both the document's versions we considered, i.e. the original TEI-based version of *The Tempest* and its modified version:

```
# The string "Ari." (original) and "Ariel" (modified)
     within the element "speaker"
:ari-string a earmark:XPathPointerRange ;
  earmark:refersTo :content ;
  earmark:hasXPathContext
    "//body/div[2]/sp[43]/speaker" .

# The string "Master"(both original and modified)
:master-string a earmark:XPathPointerRange ;
  earmark:refersTo :content ;
  earmark:hasXPathContext "//body/div[2]/sp[43]/ab" ;
  earmark:begins "17"^^xsd:nonNegativeInteger ;
  earmark:ends "23"^^xsd:nonNegativeInteger .

# The other two ranges are defined similarly
```

To persuade us of the effectiveness of EARMARK in managing documents that change over time, we can consider the following example:

```
Docuverse modified = ed.createURIDocuverse("content",
  URI.create("http://www.essepuntato.it/2013/dhcase/
       thetempest-modified.xml"));
Range ari_string = ed.createXPathPointerRange(
  "ari-string", modified, null, null,
  "//body/div[2]/sp[43]/speaker");
ed.appendChild(ari_string);
Range master_string = ed.createXPathPointerRange(
  "master-string", modified, 17, 23,
  "//body/div[2]/sp[43]/ab");
ed.appendChild(master_string);
```

As shown in the above excerpt, the EARMARK API implements clever mechanisms to calculate the string described by ranges when missing some information such as the starting and the ending locations (or both). In particular, when the starting/ending location is missing, the API implicitly consider the lowest/highest possible location within the specified context – which can be either the whole docuverse content or some restriction of it according to the XPath context of an XPath pointer range. For instance, for the range *:ari-string* we did not specify both begin and end locations so as to enable the API to take the entire text within the element *speaker* regardless its length.

4.2 Annotators vs. authors

In the previous discussions, we always referred to the annotator as a person different from the author of the document. Most of the times, this is not completely true. Authors do annotate their bare documents: they routinely add meta information, details about the structure and presentational hints (for example the `rend='italic'` attributes in the except in figure 1).

There exist various generic ways to annotate documents, especially XML documents: RDFa, microdata, microformats, etc. RDFa uses specific attributes to create relations between entities and pieces of the document, for instance to state that a certain element contains the `foaf:name` of an entity. Microdata and microformats are HTML-only techniques with a similar role: they are used to provide data about a resource through the text that has been marked in the document (ab)using certain features of HTML.

All these technologies are easy to use when the information to be annotated fits the structure and limits of RDF data properties. It is hard, instead, to describe more accurate annotations, for example it is not possible to distinguish between stating that a property has a certain value represented by a string that is part of the document and stating that a property as as its value exactly a certain string in a certain position in the document, in other words stating "the first speaker identifier is 'Ari.'" versus stating "the first speaker identifier is exactly the string 'Ari.' that appears in the first `sp` element". This kind of annotation is employed, for example, in literary criticism to describe slight but meaningful variations of the spelling of a word though a manuscript.

EARMARK allows the author of the markup of a document to manage both embedded and stand-off markup upon the same document. For instance, one can define in TEI only one of the hierarchies shown in figure 2 and, thus, create the other as an EARMARK stand-off assertion. For instance, let us take into account the following excerpt (another modified version of *The Tempest* containing markup about speeches without lines, used as docuverse for our ranges)[9]:

```
<sp>
  <speaker rend="italic">Ari.</speaker>
  <ab>
    All haile, great Master, graue Sir, haile: I come
    To answer thy best pleasure; be't to fly,
    To swim, to diue into the fire: to ride
    On the curld clowds: to thy strong bidding, taske
    Ariel, and all his Qualitie.</ab>
</sp>
```

With EARMARK and by means of the *Collections Ontology* [6], we can define the additional hierarchy of lines by adding appropriate EARMARK elements and ranges using the usual docuverse in the following way:

```
# Content of the first line
:l1-range a earmark:XPathPointerRange ;
  earmark:refersTo :content ;
  earmark:hasXPathContext "//body/div[2]/sp[43]/ab" ;
  earmark:begins "0"^^xsd:nonNegativeInteger ;
  earmark:ends "49"^^xsd:nonNegativeInteger .

# Content of the second line
:l2-range a earmark:XPathPointerRange ; ...

# The element 'body'
:body a earmark:Element, co:List ;
  earmark:hasGeneralIdentifier "body"^^xsd:string ;
  earmark:hasNamespace
    "http://www.tei-c.org/ns/1.0"^^xsd:anyURI ;
  co:firstItem [ a co:ListItem ; co:itemContent ... ;
  co:nextItem [ ...
  co:nextItem [ a co:ListItem ; co:itemContent :l1 ;
  co:nextItem [ a co:ListItem ; co:itemContent :l2 ;
  ... ] ] ... ] ] .

# The first element 'l'
:l1 a earmark:Element, co:List ;
  earmark:hasGeneralIdentifier "l"^^xsd:string ;
  earmark:hasNamespace
    "http://www.tei-c.org/ns/1.0"^^xsd:anyURI ;
  co:firstItem [ a co:ListItem ;
    co:itemContent :l1-range ] .

# The second element 'l'
:l2 a earmark:Element, co:List ; ...
```

Note that, even considering this modifications, the XPath pointer ranges defined previously (i.e. *:ari-string*, *:ariel-string*, *:spirit-string* and *:master-string*) and, thus, the related annotation on those ranges still apply correctly. This example can be realized in a trivial way using the EARMARK API, as shown in the following example:

[9]The modified document is available at `http://www.essepuntato.it/2013/dhcase/thetempest-modified-another.xml`.

```
Docuverse doc = ed.createURIDocuverse(
    "content", URI.create(
    "http://www.essepuntato.it/2013/dhcase/thetempest
        -modified-another.xml"));
Element body = ed.createElement("body", URI.create(
  "http://www.tei-c.org/ns/1.0"), Type.List);
ed.appendChild(body);
...
Element l1 = ed.createElement("l1", URI.create(
  "http://www.tei-c.org/ns/1.0"), Type.List);
body.appendChild(l1);
Range l1_range = ed.createXPathPointerRange(
  "l1-range", doc2, 0, 49,
  "//body/div[2]/sp[43]/ab");
l1.add(l1_range);
Element l2 = ...
```

5. HAPPILY EVER AFTER: FROM FAIRY TALES TO THE COLD XML REALITY

The approach we propose in the previous sections solves many of the problems related to the activityies of annotating documents. However this approach uses tools and technologies of the Semantic Web while the majority of applications, editors and visualization tools are still based on XML. For this reason we developed a tool called FRETTA (From EARMARK To Tag) [1] that can embed arbitrary EARMARK annotations (including non-sequential, non-hierarchical and non-contiguous ones) inside XML documents.

The main problems addressed by FRETTA are, first, the preservation of both the overlapping hierarchies (over the same content) expressed through the structural markup and, second, the correct embedding of the assertions.

With regard to the first issue, various communities have developed techniques or "tricks" [8] to work around the limitations imposed by hierarchical structure of XML: for example, the TEI guidelines [7] describe methods that can be used to express multiple hierarchies over the same content as a single XML hierarchy. Among these techniques there are *milestones* (i.e. overlapping structures are expressed through a pair of empty elements to mark the boundaries of the "content") and *fragmentation* (i.e. overlapping structures are split into individual, non-overlapping elements that are linked through id-idref pairs). FRETTA combines into a single Java framework a half dozen of syntactic techniques: users that need to convert documents from EARMARK to XML can decide which tricks to use in each format (e.g., milestones for line elements when going to TEI and wrappers going to HTML) and how EARMARK elements can be encoded in that format. FRETTA then performs the conversion taking care of the remaining details.

In addition to knowing many such tricks, FRETTA also knows different embedding mechanisms to deal with the annotations contained in EARMARK documents. For example, it is possible to choose to inject the annotations into XML attributes using RDFa, or to use a solution based on stand-off where the annotations are kept in external documents.Thus, not only FRETTA allows to convert documents from EARMARK to XML preserving all the original information, but it is also able to carry out this process in a customizable and flexible manner.

There is a further aspect of FRETTA worth highlighting. All markup elements and fragments are identified by URIs within EARMARK documents. This makes it possible to refer to content belonging to other EARMARK documents, without redefining the markup and the text of the elements. These references can be exploited to create simple inclusions but also to create *permanent* and *live* connections between the inclusion and the original source of content, and to build sophisticated applications on top of these connections.

This is a form of *transclusion*, similar to what has originally been proposed by Ted Nelson for the Xanadu project [18]. In a transclusion, the included content is never actually copied but stored as a virtual reference to the original location. The goal of the Xanadu project was to build a global workspace where all users could freely reuse, comment on and link to any piece of content. It is interesting to note that the current EARMARK model provides everything is needed to express xanalogical relations between documents, and to mix them with relations to external entities. The following example, for instance, shows how to transclude the first line spoken by Ariel from the previous excerpts into an arbitrary document:

```
# An EARMARK document for the following structure:
# <div>
#   <p>
#     This document quote, as follows,
#     the first line of Ariel speech
#     defined in the excerpts of another
#     EARMARK document:</p>
#   <blockquote>
#     <l>All haile, great Master, graue Sir,
#   haile: I come</l>
#   </blockquote>
# </div>

<http://www.essepuntato.it/another-example>
  a owl:Ontology .

another:content a earmark:StringDocuverse ;
  earmark:hasContent "This document quote,
    as follows, the first line of Ariel speech
    defined in the excerpts of another EARMARK
    document:"^^xsd:string .

another:range a earmark:PointerRange ;
  earmark:refersTo another:content .

another:div a earmark:Element, co:List ;
  earmark:hasGeneralIdentifier "div"^^xsd:string ;
  co:firstItem [ a co:ListItem ;
    co:itemContent another:p ;
  co:nextItem [ a co:ListItem ;
    co:itemContent another:blockquote ] ] .

another:p a earmark:Element, co:List ;
  earmark:hasGeneralIdentifier "p"^^xsd:string ;
  co:firstItem [ a co:ListItem ;
    co:itemContent another:range ] .

another:blockquote a earmark:Element, co:List ;
  earmark:hasGeneralIdentifier
    "blockquote"^^xsd:string ;
  co:firstItem [ a co:ListItem ; co:itemContent :l1].

# We declare where we have to look for :l1
:l1 a earmark:Element ;
  rdfs:isDefinedBy
    <http://www.essepuntato.it/example> .
```

Implementing this example using the EARMARK API is straightforward:

```
String another = ... /* the same as Turtle prefix */
EARMARKDocument ed = new EARMARKDocument(URI.create(
  "http://www.essepuntato.it/another-example"));
Docuverse doc = ed.createStringDocuverse(
  "This document quote, as follows, ...");
Element div = ed.createElement(
  another + "div", "div", Type.List);
Element p = ed.createElement(
  another + "p", "p", Type.List);
Range range = ed.createPointerRange(
  another + "range", doc, null, null);
```

```
Element blockquote = ed.createElement(
  another + "blockquote", "blockquote", Type.List);
Element l1 = ed.createElement(
  ex + "l1", "l", Type.List);

div.appendChild(p);
p.appendChild(range);
div.appendChild(blockquote);
blockquote.appendChild(l1);

Model model = ed.getModel();
model.add(l1, RDFS.isDefinedBy, model.createResource(
  "http://www.essepuntato.it/example"));
```

There is no direct support for transcluding content in the current API but there will be in future releases. The full implementation of a xanalogical system opens several tricky issues in terms of content merging, synchronization, and content loading strategies that we do not need to discuss here. What is relevant is the fact that EARMARK opens the possibility of integrating transclusions with Semantic Web technologies and data sources. To the best of our knowledge, such a research direction has never been explored but we believe it could lead to fascinating results and applications.

6. REFERENCES

[1] Barabucci, G., Peroni, S., Poggi, F., and Vitali, F. Embedding semantic annotations within texts: the fretta approach. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2012), SAC '12, ACM, pp. 658–663.

[2] Bizer, C., Heath, T., and Berners-Lee, T. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst. 5*, 3 (2009), 1–22.

[3] Brickley, D., and Miller, L. FOAF vocabulary specification 0.98. Tech. rep., Feb. 2010. `http://xmlns.com/foaf/spec/20100809.html`. Latest version available at `http://xmlns.com/foaf/spec/`.

[4] Ciccarese, P., Ocana, M., and Clark, T. Open semantic annotation of scientific publications using domeo. *Journal of Biomedical Semantics 3*, 1 (2012), 1–14.

[5] Ciccarese, P., Ocana, M., Garcia Castro, L., Das, S., and Clark, T. An open annotation ontology for science on web 3.0. *Journal of Biomedical Semantics 2*, 2 (2011), 1–24.

[6] Ciccarese, P., and Peroni, S. The collections ontology: creating and handling collections in owl 2 dl frameworks. To appear in Semantic Web: Interoperability, Usability, Applicability., 2013.

[7] Consortium, T. Tei p5: Guidelines for electronic text encoding and interchange.

[8] DeRose, S. J. Markup overlap: A review and a horse. In *Proceedings of the Extreme Markup Languages® 2004 Conference, 2-6 August 2004, Montreal, Quebec, Canada* (2004).

[9] Di Iorio, A., Peroni, S., and Vitali, F. A semantic web approach to everyday overlapping markup. *JASIST 62*, 9 (2011), 1696–1716.

[10] Di Iorio, A., Peroni, S., and Vitali, F. Using semantic web technologies for analysis and validation of structural markup. *Int. J. Web Eng. Technol. 6*, 4 (2011), 375–398.

[11] Hellmann, S., Lehmann, J., and Auer, S. Linked-data aware uri schemes for referencing text fragments. In *EKAW* (2012), pp. 175–184.

[12] Huitfeldt, C., and Sperberg-McQueen, C. M. Texmecs: An experimental markup meta-language for complex documents. Working paper of the project MLCD, University of Bergen., 2001.

[13] ISO. *ISO 24612:2012 Language resource management — Linguistic annotation framework (LAF)*. ISO, 2012.

[14] Lebo, T., Sahoo, S., and McGuinness, D. PROV-O: The PROV Ontology. Recommendation, W3C, Apr. 2013. `http://www.w3.org/TR/2013/REC-prov-o-20130430/`.

[15] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal* (2013). Under review.

[16] Marinelli, P., Vitali, F., and Zacchiroli, S. Towards the unification of formats for overlapping markup. *The New Review of Hypermedia and Multimedia 14*, 1 (2008), 57–94.

[17] McCarron, S. XHTML+RDFa 1.1. Recommendation, W3C, June 2012. `http://www.w3.org/TR/2012/REC-xhtml-rdfa-20120607/`.

[18] Nelson, T. *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, 1980.

[19] Peroni, S. Earmark ontology. `http://www.essepuntato.it/2008/12/earmark`.

[20] Peroni, S., Gangemi, A., and Vitali, F. Dealing with markup semantics. In *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011* (2011), C. Ghidini, A.-C. N. Ngomo, S. N. Lindstaedt, and T. Pellegrini, Eds., pp. 111–118.

[21] Picca, D., Gliozzo, A. M., and Gangemi, A. Lmm: an owl-dl metamodel to represent heterogeneous lexical knowledge. In *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco* (2008).

[22] Sanderson, R., Ciccarese, P., and de Sompel, H. V. Open annotation data model. Community draft, W3C, Feb. 2013. `http://www.openannotation.org/spec/core/20130208/`.