A Semantic Web Approach To Everyday Overlapping Markup

Angelo Di Iorio, diiorio@cs.unibo.it, Department of Computer Science, University of Bologna
Silvio Peroni, speroni@cs.unibo.it, Department of Computer Science, University of Bologna
Fabio Vitali, fabio@cs.unibo.it, Department of Computer Science, University of Bologna

Overlapping structures in XML are not the symptoms of a misunderstanding of the intrinsic characteristics of a text document, nor the evidence of extreme scholarly requirements far beyond those needed by the most common XML-based applications. On the contrary, overlaps have started to appear in a large number of incredibly popular applications hidden under the guise of syntactical tricks to the basic hierarchy of the XML data format. Unfortunately, syntactical tricks have the drawback that the affected structures require complicated workarounds to support even the simplest query or usage.

In this paper we present EARMARK, an approach to overlapping markup that simplifies and streamlines the management of multiple hierarchies on the same content, and provides an approach to sophisticated queries and usages over such structures without the need of ad-hoc applications, simply by using Semantic Web tools and languages.

We compare how relevant tasks (e.g., the identification of the contribution of an author in a Word Processor document) are of some substantial complexity when using the original data format, and become more or less trivial when using EARMARK. We finally evaluate positively the memory and disk requirements of EARMARK documents in comparison to Open Office and Microsoft Word XML-based formats.

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

Additional Key Words and Phrases: EARMARK, OWL, RDF, XML, overlapping

1. INTRODUCTION

The overwhelming consensus among XML practitioners is that documents are trees, the hierarchy is the fundamental data structure, and violations of the hierarchy are errors or unnecessary complications. Therefore, overlapping markup has received ambivalent, almost schizoid considerations in the field of markup languages. Traditionally, overlaps were the hallmarks of bad HTML coders and naive HTML page editors, taking advantage of the unjustified benevolence in web browsers that would display basically any HTML regardless of proper nesting. At the same time, far from the awareness of the general public, overlaps have been a fringe, almost esoteric discipline of scholars in the humanities, competently used for arcane specifications of linguistic annotations and literary analysis.

But although the first type of overlap was judged with scorn and the second with awe, they both fundamentally represent a situation that is more common than thought, and the scholars were only more aware, and not more justified, about the need to represent overlaps.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

Generally, overlap is needed when multiple independent items refer to the same segment, either when considering textual markup documents or multimedia structures [Salembier & Benitez 2007]. As regards to documents with markup, we need overlap whenever multiple markup elements needs to be applied over the same content, and these elements happen to be independent of each other. In some (rather frequent) situations, this independence means that the content referred to by some elements is partially but not completely the same as the content referred to by other elements.

This situation is more frequent than may appear: not only bad HTML code and arcane linguistic annotations use overlap, but many more mainstream and mundane examples exist. For instance, change tracking in a office document is often at odds with the underlying structure of the text; microformats [Allsopp 2007] and RDFa [Adida et al. 2008] annotations may need to refer to concepts that span across multiple XML elements; complex data structures (e.g. biological data) force graphs into trees and hide multiple parentage as internal references. And the list could go on.

Differently from SGML, that is able to handle some overlapping scenarios through the CONCUR notation [Goldfarb 1990], XML grammatically imposes and requires a strict hierarchy of containment generating a single mathematical tree of the document where no overlap is allowed. This requirement has been turned into an intrinsic characteristic of the documents XML was meant to represent, rather than a syntactical and conceptual constraint into which these documents need to fit. Thus, whenever authors needed to cope with independent markup elements, they managed either by naively ignoring the hierarchical limitation (and therefore creating invalid documents), or by creating careful workarounds within the syntactical constraint, or even by inventing completely new markup languages that allow some types of overlap. But while new multi-hierarchical markup languages such as TexMecs [Huitfeldt & Sperberg-McQueen 2003] and LMNL [Tennison & Piez 2002] have a small number of adepts and applications, and while bad HTML coders and bad HTML page editors are disappearing from the market, the careful workarounds within the XML syntax [TEI Consortium 2005], such as segmentation, milestones or standoff markup, are to this day frequently used and ubiquitous.

All workarounds share the same approach of hiding structural information about a secondary hierarchy under the guise of something else: split individual elements, empty boundary elements, indirect references, etc. The result is that the secondary structural information is hidden or its importance is lessened, so as not to break or obfuscate the main hierarchy expressed in the visible XML structure. But this comes at a price: structures specified through workarounds are more difficult to find, identify and act upon than the structures in the main XML hierarchy. Thus, trivial searches that should amount to a short XPath in a more direct situation end up being multiple lines long, pretty basic visualizations require incredibly complex XSLT stylesheets, specific choices of the main markup hierarchy actually prevent some features of the secondary markup to even exist, etc. So, although workarounds exist and can be used, hierarchies expressed through them are second class citizens that cannot fully exploit the sophisticated tools that the XML language provides.

In this paper we show how EARMARK (Extremely Annotational RDF Markup), our proposal for managing overlapping markup, does not generate first and second class hierarchies, and allows existing, sophisticated tools to be used on all markup even in the presence of overlaps. Rather than creating a completely new language requiring completely new tools and competencies, EARMARK uses Semantic Web technologies and Semantic Web tools to obtain many of the results obtainable with usual XML tools.

EARMARK defines markup vocabularies by means of OWL ontologies [W3C OWL Working Group 2009]. Since each individual markup item is an independent assertion over some content or some other assertions, overlaps of content is not a problem as well

as all the issues connected to physical embedding and containments, such as contiguity and document order. Furthermore, by using standard Semantic Web technologies, fairly sophisticated functionalities can be provided over EARMARK documents.

Through EARMARK, operations that were previously very hard or impossible exactly because of the interferences of the multiple hierarchies or of the workarounds they employed, become now fundamentally trivial, since no syntactical tricks are employed and the different hierarchies do not interfere with each other. Thus for instance identifying the individual contributions in a multi-authored MS Word or Open Office document is quite hard on their original XML formats, and becomes trivial when the same documents are converted into EARMARK.

This paper is an extended version of previous works on EARMARK ([Peroni & Vitali 2009] and [Di Iorio et al. 2010]). In those papers we focused on identifying workarounds for overlapping data existing in real XML documents and translating them into EARMARK assertions. In those papers we also sketched the EARMARK ontology and presented a simple implementation of EARMARK-aware tools. This paper follows and extends them and provides some novel contributions:

— The systematic analysis of the EARMARK model, with particular attention to data typing and overlapping structures.
— The discussion of further applications for the ontological EARMARK approach. In particular, we show how EARMARK can be used to improve the content filtering and reversions mechanisms of wikis.
— The brief description of a process – called ROCCO – for generating EARMARK documents from existing XML documents (even ones that use workarounds for overlapping structures)
— An evaluation of EARMARK efficiency when dealing with multiple hierarchies in comparison with the XML structures used by popular XML-based formats such as Open Office and MS Word.

The paper is structured as follows: in Section 2 we provide a brief overview of existing approaches to overlap using workarounds in XML or ad-hoc markup metalanguages, and in Section 3 we provide a few examples of situations where overlaps are used today and sometimes in rather mainstream situations. In Section 4 we present the EARMARK model and its rules. In Section 5 we discuss some use-cases that are meant to demonstrate the superiority of the EARMARK approach to a traditional XML format, especially when overlaps come into question. Section 6 goes into the details of the generation of EARMARK documents, converting legacy documents. Section 7 contains an initial evaluation of the efficiency of EARMARK compared to popular XML based data formats such as ODT ad OOXML, leading to our conclusions in Section 8.

2. EXISTING APPROACHES TO OVERLAPPING

The need for multiple overlapping structures over documents using markup syntaxes such as XML and SGML is an age-old issue, and a large amount of literature exists about techniques, languages and tools that allow users to create multiple entangled hierarchies over the same content. A good review can be found in [DeRose 2004].

Some of such research proposes to use plain hierarchical markup (i.e., XML) and employ specially tailored elements or attributes to express the semantics of overlapping in an implicit way. The TEI Guidelines [TEI Consortium 2005] present a number of different techniques that use SGML/XML constructs to force multiple hierarchies into a single one, including:

— *milestones* (the overlapping structures are expressed through empty elements to mark the boundaries of the “content”),

— *fragmentation* (the overlapping structures are split into individual non-overlapping elements that may even be linked through *id-idref* pairs) and
— *standoff markup* (the overlapping structures are placed elsewhere and indirectly refer to their would-be locations through pointers, locators and/or *id-idref* pairs).

Given the large number of techniques to deal with overlapping structures in XML, in [Marinelli et al. 2008] we presented a number of algorithms to convert XML documents with overlapping structures from and to the most common approaches, as well as a prototype implementation.

In [Riggs 2002] the author introduces a slightly different technique for fragmentation within XML structures. In this proposal, *floating elements*, i.e., those elements that do not fall in a proper or meaningful hierarchical order, are created using the name of the element followed by an index referring to its semantically-related parent element. For example, the floating element `<name.person[2]>John</name.person[2]>` means that `<name>John</name>` is semantically child of the second occurrence of the element *person*, even though the floating element is not structurally contained by its logical parent.

Other research even proposes to get rid of the theory of trees at the base of XML/SGML altogether, and use different underlying models and newly invented XML-like languages that allow the expression of overlaps through some kind of syntactical flourishing.

For instance, *GODDAG* [Sperberg-McQueen & Huitfeldt 2004] is a family of graph-theoretical data structures to handle overlapping markup. A GODDAG is a Direct Acyclic Graph whose nodes represent markup elements and text. Arcs are used to explicitly represent containment and father-child relations. Since multiple arcs can be directed to the same node, overlapping structures can be straightforwardly represented in GODDAG. Full GODDAGs cannot be linearized in any form using embedded markup, but *restricted GODDAGs*, a subset thereof, can be and has been linearized into TexMecs [Huitfeldt & Sperberg-McQueen 2003], a multi-hierarchical markup language that also allows full GODDAGs through appropriate non-embedding workarounds, such as standoff markup.

LMNL [Tennison & Piez 2002] is a general data model based on the idea of *layered text fragments and ranges*, where multiple types of overlap can be modeled using concepts drawn from the mathematical theory of intervals. Multiple serializations of LMNL exists, such as CLIX and LMNL-syntax.

XConcur [Schonefeld & Witt 2006] is a similar solution based on the representation of multiple hierarchies within the same document through *layers*. Strictly related to its predecessor CONCUR as it was included in the SGML, XConcur was developed in conjunction with the validation language XConcur-CL to handle relationships and constraints between multiple hierarchies.

The *variant graph* approach [Schmidt & Colomb 2009] is also based on graph theory. Developed to deal with textual variations – that generate multiple versions of the same document with multiple overlapping hierarchies – this theory proposes a new data model to represent literary documents and a graph linearization (based on lists) that scales well even with a large number of versions. The same authors recently presented an extension of their theory that also allows users to merge multiple variants into one document [Schmidt 2009]. In [Portier & Calabretto 2009] a detailed survey about overlapping approaches was presented, also discussing the MultiX data model – that uses W3C standard languages such as XInclude to link and fetch text fragments within overlapping structures – and a prototype editor for the creation of multi-structured documents.

In [Tummarello et al. 2005] a proposal for using RDF as a standoff notation for overlapping structures of XML documents was proposed. Since this proposal has many affinities with the one we are presenting in this paper, in section Section 4.5 we discuss in deep its characteristics and compare it with ours.

3. MORE FREQUENT THAN ONE MAY THINK: OVERLAPPING IN THE WILD

Overlapping structures have been considered often as appropriate only in highly specific contexts and basically for scholars: the solutions proposed in literature were complex since they were considered grounded in the intrinsic complexity of the topics themselves. Yet, overlapping structures can be found in many more fields than these, and even mainstream applications generate and use markup with overlapping structures. While the complexity of overlapping is hidden to the final user, application that consume such data may very well find it rather difficult to handle such information. In the following we discuss three very different contexts where overlapping already exist and fairly relevant information is encoded in multiple independent structures, leaving to special code the task of managing the complexity.

3.1. Change Tracking in Office Document Format

Word processors such as Microsoft Word and Open Office provide users with powerful tools for tracking changes, allowing each individual modification by individual authors to be identified, highlighted, and acted upon (e.g. by accepting or discarding them). The intuitiveness of the relevant interfaces actually hides the complexity of the data format and of the algorithms necessary to handle such information.

For instance, the standard ODT format [JTC1/SC34 WG 6. 2006] used by Open Office, when saving change tracking information, relies on two specific constructs for insertions and deletions that may overlap with the structural markup. While adding a few words within a paragraph is not in itself complex, as it does not require the breaking of the fundamental structural hierarchy, conversely changes that affect the structure itself (e.g. the split of one paragraph in two by the insertion of a return character, or vice versa the joining of two paragraphs by the elimination of the intermediate return character) require that annotations are associated to the end of a paragraph and the beginning of the next, in an unavoidably overlapping pattern. ODT uses milestones and standoff markup for insertions and deletions respectively, and also relies on standoff markup for annotations about the authorship and date of the change.

For instance, the insertion of a return character and a few characters in a paragraph creates a structure as follows:

```
<text:tracked-changes><text:changed-region text:id="S1">
  <text:insertion><office:change-info>
    <dc:creator>John Smith</dc:creator>
    <dc:date>2009-10-27T18:45:00</dc:date>
  </office:change-info></text:insertion></text:changed-region>
  [... other changes ...]
</text:tracked-changes>
[... content ...]
<text:p>The beginning and
  <text:change-start text:change-id="S1"/></text:p>
<text:p> also<text:change-end text:change-id="S1"/> the end.</text:p>
```

The empty elements `<text:change-start/>` and `<text:change-end/>` are *milestones* marking respectively the beginning and the end of the range that constituted the insertion, while the element `<text:insertion>`, before the beginning of the document content, is *standoff markup* for the metadata about the change (author and date information).

Similarly, a deletion creates a structure as follows:

```
<text:tracked-changes><text:changed-region text:id="S2">
  <text:deletion><office:change-info>
    <dc:creator>John Smith</dc:creator>
    <dc:date>2009-10-27T18:46:00</dc:date>
  </office:change-info><text:p/><text:p/></text:deletion>
  </text:changed-region>
  [... other changes ...]
</text:tracked-changes>
[... content ...]
<text:p>The beginning and
  <text:change text:change-id="S2" />also the end.</text:p>
```

The element <text:change/> represents a *milestone* of the location where the deletion took place in the content, and the corresponding *standoff markup* annotation <text:deletion> contains not only the metadata about the change, but also the text that was deleted.

The OOXML format [JTC1/SC34 WG 4. 2008] (the XML-based format used by Microsoft Office 2007 and standardized by ISO in 2008), on the other hand, uses a form of *segmentation* to store change-tracking information across all previous elements involved.

```
<w:p>
  <w:pPr><w:rPr>
    <w:ins w:id="0" w:author="John Smith"
      w:date="2009-10-27T18:50:00Z"/>
  </w:rPr></w:pPr>
  <w:r><w:t>The beginning and </w:t></w:r></w:p>
<w:p>
  <w:ins w:id="1" w:author="John Smith"
    w:date="2009-10-27T18:50:00Z">
    <w:r><w:t>also </w:t></w:r></w:ins>
  <w:r><w:t>the end.</w:t></w:r></w:p>
```

This heavily simplified version of an OOXML document shows two separate changes: the first is the insertion of a return character and the second is the insertion of a word. These modifications are not considered as a single change, and therefore the segments are not connected to each other, but simply created as needed to fit the underlying structure.

In fact, change tracking in OOXML is a fairly complex proposition. Although providing more complete coverage of special cases and situations than ODT, dealing with its intricacies is not for the casual programmer. Even a simple XSLT stylesheet to show inserted text in a different color and hide deleted text may run several hundred lines of code[1].

3.2. Overlapping with Microformats

Microformats [Allsopp 2007] add semantic markup to web documents by using common structures of the HTML language itself, in particular the *class* attribute.

The HTML code is annotated using microformats so as to provide new semantic, machine-processable assertions. In the following example, a plain HTML table is enriched with metadata about events[2] and people[3]:

[1] http://OOXMLdeveloper.org/archive/2006/09/07/625.aspx
[2] HCalendar, http://microformats.org/wiki/hcalendar
[3] HCard, http://microformats.org/wiki/hcard

```
<body>
  <p class="vevent">
    <span class="summary">WWW 2010 Conference</span>:
    <abbr class="dtstart" title="2010-04-26">April 26</abbr>
    -<abbr class="dtend" title="2010-10-30">30</abbr>,
    <span class="location">Raleigh, NC, USA</span>.</p>
  <table>
    <tr><th>Name</th><th>Role</th></tr>
    <tr class="vcard">
      <td class="fn">Juliana Freire</td>
      <td class="role">Program Committee Chair</td></tr>
    <tr class="vcard">
      <td class="fn">Michal Rappa</td>
      <td class="role">Conference Chair</td></tr>
    <tr class="vcard">
      <td class="fn">Paul Jones</td>
      <td class="role">Conference Chair</td></tr>
    <tr class="vcard">
      <td class="fv">Soumen Chakrabarti</td>
      <td class="role">Program Committee Chair</td></tr>
  </table>
<body>
```

The table was enriched by additional data declaring it to be an event (a conference) and data about the event itself – the url, summary, location – and about four relevant individuals – with their names and roles within the conference – were associated where necessary to the actual content of the table.

So far, so good, and no overlap to speak about. Things change dramatically, though, when the overall structure of the main hierarchy (the HTML table) is at odds with the intrinsic hierarchy of the microformat data, for instance if the people are organized in columns rather than rows. For instance:

```
<table>
  <tr>
    <td>Program Committee Chair</td><td>Conference Chair</td>
    <td>Conference Chair</td><td>Program Committee Chair</td></tr>
  <tr>
    <td>Juliana Freire</td><td>Michael Rappa</td>
    <td>Paul Jones</td><td>Soumen Chakrabarti</td></tr>
</table>
```

Unfortunately, vcards are a hierarchy themselves, and if the hierarchy of vcards is organized differently from the hierarchy of the HTML table, as in the latter case, it is just impossible to define the four vcards for the four people organizing the conference. Thus in plain HTML the choice of one of two possible presentation models for the main hierarchy of content makes it trivial or completely impossible the existence of the second hierarchy.

A possible and partial solution to express vcard hierarchies in the latter example is RDFa [Adida et al. 2008], a W3C recommendation. It describes a mechanism to embed RDF statements into HTML documents by using some HTML attributes (*href*, *rel*, *rev*, *content*) in combination with other *ad hoc* attributes (*property*, *about*, *typeof*) proposed in the recommendation itself.

```
<table
  xmlns:vc="http://www.w3.org/2006/vcard/ns#"
  xmlns:my="http://www.essepuntato.it/2010/05/myVCard#">
  <tr>
```

```
    <td about="my:pcc" typeof="vc:Role">
      Program Committee Chair</td>
    <td about="my:cc" typeof="vc:Role">Conference Chair</td>
    <td about="my:cc" property="vc:hasName">
      Conference Chair</td>
    <td about="my:pcc" property="vc:hasName">
      Program Committee Chair</td></tr>
  <tr>
    <td about="my:jf" rel="vc:role" resource="my:pcc">
      <span property="vc:fn">Juliana Freire</span></td>
    <td about="my:mr" rel="vc:role" resource="my:cc">
      <span property="vc:fn">Michael Rappa</span></td>
    <td about="my:pj" rel="vc:role" resource="my:cc">
      <span property="vc:fn">Paul Jones</span></td>
    <td about="my:sc" rel="vc:role" resource="my:pcc">
      <span property="vc:fn">Soumen Chakrabarti</span></td></tr>
</table>
```

Since all attributes live in the context of elements, the price to pay is that to assert everything we want to assert we often need to add some structurally unnecessary elements to the current markup hierarchy of a document, needed only to add the RDF statements (e.g., the *span* elements emphasized above). Even if that does not represent a significant problem for strict Semantic Web theorists, document architects and markup expert see this as a kludge and an inelegant compromise.

3.3. Wikis: no overlapping where some should be

The strength of wikis lies in their allowing users to modify content at any time. The mechanisms of change-tracking and rollback that are characteristics of all wikis, in fact, promote users' contributions and make "malicious attacks" pointless in the long run, since previous versions can be easily restored.

A number of tools exist that automatically discover "wiki vandalisms" and provide users with powerful interfaces to surf changes, *diff* subsequent versions and revert content. For instance, Huggle[4] is an application dealing with vandalism in Wikipedia, based on a proxy architecture and .NET technologies. A straightforward interface allows users to access any version of a page, highlights contributions of a specific user and reverts the content to old versions.

Even client-side tools – meant to be installed as browsers extensions or bookmarklets – exist to extend the rollback mechanisms of Wikipedia, giving users more flexibility and control over (vandalistic) changes. For instance, Lupin[5] is a set of javascript scripts that check a wiki page against a list of forbidden terms so that authors can identify undesirable modifications and restore previous (good) versions without a continuous control over the full content of the page; yet again, Twinkle[6] provides users powerful rollback functions and includes a full library of batch deletion functions, automatic reporting of vandals, and users notification functions.

These tools are successful in highlighting vandalism and in identifying versions created by malicious users. However, although it is possible to revert the page to any previous version, all changes (even acceptable ones) that were subsequent to the malicious version cannot be automatically inherited by the restored page.

For instance, let us consider versions V1, V2, and V3 of a wiki page, where versions V1 contains a baseline (acceptable) content, V2 is identified as a partial vandalism

[4]http://en.wikipedia.org/wiki/Wikipedia:Huggle
[5]http://en.wikipedia.org/wiki/User:Lupin/Anti-vandal_tool
[6]http://en.wikipedia.org/wiki/Wikipedia:Twinkle

and is agreed to be removed, but V3 contains (possibly, in a completely different section than the target of the malicious attack) relevant and useful content that was added before the vandalistic version V2 was declared as such. The task of removing the modifications of version V2 while maintaining (whatever is possible of) version V3 is a difficult, error-prone and time-consuming task if done manually, yet there is no tool we are aware of that automatically filters contributions from multiple versions and merges them into a new one (or, equivalently, removes only selected intermediate versions).

Yet, it is possible to characterize the interdependencies between subsequent changes to a document in a theoretical way. In fact, literature has existed for a long time on exactly these themes (see for instance [Durand 2008] [Durand 1994]). Although a detailed discussion of abstract models of interconnected changes is out of scope for this paper – details and authoritative references can be found in the above mentioned works – what is relevant in this discussion is that they happen to assume a hierarchical form that is frequently at odds with the hierarchical structure of the content of the document, and as such most issues derive from the data structures in which content is stored and from the model for manipulating these structures. For instance, the fact that in the wiki perspective each version is an independent unit that shares no content (even unchanged content) with the other versions prevents considering multiple versions as overlapping structures coexisting on the same document. If we were able to make these hierarchies explicit we would be able to create models and tools to manipulate these documents in a more powerful way and to exploit the existing interconnections between the overlapping hierarchies.

4. INTRODUCTION TO EARMARK AND ITS SUPPORT FOR OVERLAPPING FEATURES

The presence of hidden overlapping structures – transparent to users but very difficult to handle by applications – is the common denominator for the scenarios described in the previous section.

More than the overlap itself – that cannot be ignored as it does exist and carries important meanings – the problem we face lies in the way applications store such overlapping structures. In the XML world, in fact, the only way to do so is through the use of (complex) workarounds that force the multiple hierarchies into one hierarchy of a XML document. That makes very tricky to perform sophisticated analysis and searches.

This section discusses a different approach to metamarkup, called EARMARK (Extremely Annotational RDF Markup) [Di Iorio et al. 2009] [Peroni & Vitali 2009] [Di Iorio et al. 2010] based on ontologies and Semantic Web technologies. The basic idea is to model EARMARK documents as collections of addressable text fragments, and to associate such text content with OWL assertions that describe structural features as well as semantic properties of (parts of) that content. As a result EARMARK allows not only documents with single hierarchies (as with XML) but also multiple overlapping hierarchies where the textual content within the markup items belongs to some hierarchies but not to others. Moreover EAMARK makes it possible to add semantic annotations to the content though assertions that may overlap with existing ones.

One of the advantages of using EARMARK is the capability to access and query documents by using well-known and widely supported tools for Semantic Web. In fact, EARMARK assertions are simply RDF assertions, while EARMARK documents are modeled through OWL ontologies. The consequence is that query languages (such as SPARQL [Garlik & Seaborne 2010]) and actual existing tools, such as Jena[7] and Pel-

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

let[8]) can be directly used to deal with even incredibly complicated overlapping structures. What is very difficult (or impossible) to do with traditional XML technologies becomes much easier with these technologies under the EARMARK approach.

In the rest of this section we give a brief overview of the EARMARK model, while in Section 5 we describe how EARMARK can be used to deal with the issues presented earlier. The model itself is defined through an OWL document[9], summarized in Fig. 1, specifying classes and relationships. We distinguish between *ghost classes* - that define the general model - and *shell classes* - that are actually used to create EARMARK instances.

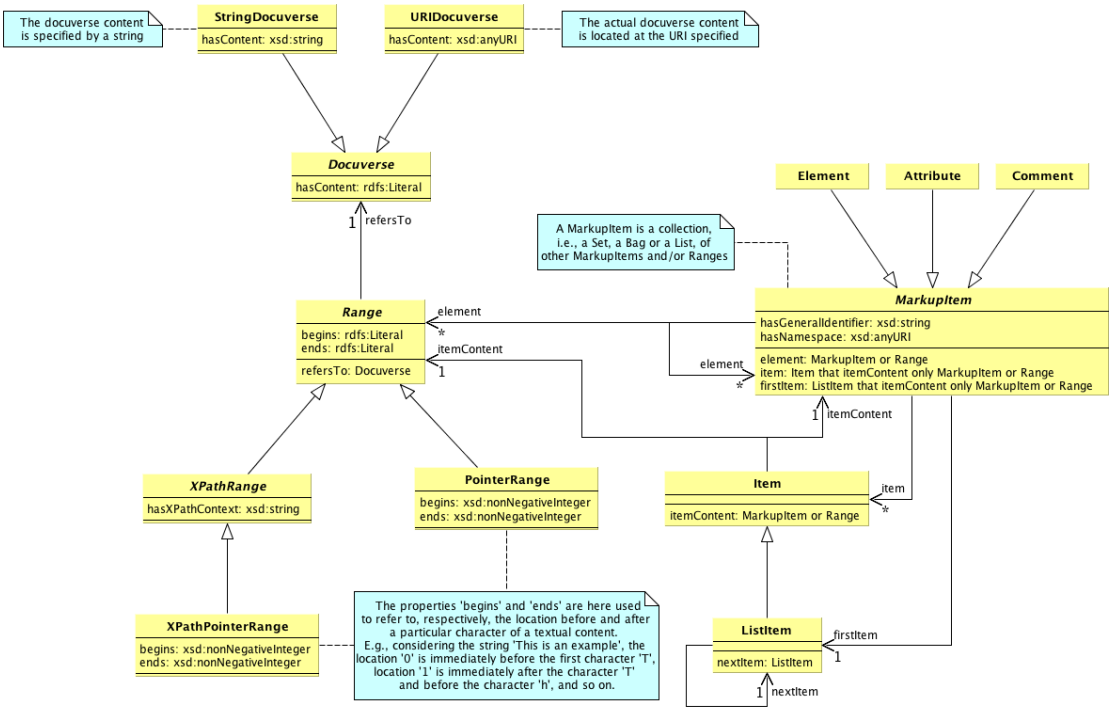

Fig. 1. An UML-like representation of the EARMARK ontology.

4.1. Ghost classes

The ghost classes describe three disjoint base concepts – *docuverses*, *ranges* and *markup items* – through three different and disjoint OWL classes[10].

[8]http://pellet.owldl.com
[9]http://www.essepuntato.it/2008/12/earmark
[10]All our OWL samples are presented using the Manchester Syntax [Horridge & Patel-Schneider 2009], which is one of the standard linearization syntaxes of OWL. The prefixes *rdfs* and *xsd* refer respectively to RDF Schema and XML Schema namespaces, while the empty prefix refers to the EARMARK ontology URI plus "#". Moreover, we use the prefix *c* to indicate entities taken from an imported ontology made for the SWAN project [Ciccarese et al. 2008], available at http://swan.mindinformatics.org/spec/1.2/collections.html.

The textual content of an EARMARK document is conceptually separated from its annotations, and is referred to through the *Docuverse* class[11]. The individuals of this class represent the object of discourse, i.e. all the containers of text of an EARMARK document.

```
Class: Docuverse

DatatypeProperty: hasContent Characteristics: FunctionalProperty
  Domain: Docuverse Range: rdfs:Literal
```

Any individual of the *Docuverse* class – commonly called a *docuverse* (lowercase to distinguish it from the class) – specifies its actual content with the property *hasContent*.

We then define the class *Range* for any text lying between two locations of a docuverse. A *range*, i.e, an individual of the class *Range*, is defined by a starting and an ending location (any literal) of a specific docuverse through the properties *begins*, *ends* and *refersTo* respectively.

```
Class: Range
  EquivalentTo:
    refersTo some Docuverse and
    begins some rdfs:Literal and
    ends some rdfs:Literal

ObjectProperty: refersTo Characteristics: FunctionalProperty
  Domain: Range Range: Docuverse

DatatypeProperty: begins Characteristics: FunctionalProperty
  Domain: Range Range: rdfs:Literal

DatatypeProperty: ends Characteristics: FunctionalProperty
  Domain: Range Range: rdfs:Literal
```

There is no restriction on locations used for the *begins* and *ends* properties. That is very useful: it allows us to define ranges that *follow* or *reverse* the text order of the docuverse they refer to. For instance, the string “desserts” can be considered both in document order, with the *begins* location lower than the *ends* location or in the opposite one, forming “stressed”[12]. Thus, the values of properties *begins* and *ends* define the way a range must be read.

The class *MarkupItem* is the superclass defining artifacts to be interpreted as markup (such as elements and attributes).

```
Class: MarkupItem
  SubClassOf:
    (c:Set that c:element only (Range or MarkupItem)) or
    (c:Bag that c:item only
      (c:itemContent only (Range or MarkupItem))

DatatypeProperty: hasGeneralIdentifier
  Characteristics: FunctionalProperty
  Domain: MarkupItem Range: xsd:string
```

[11]This class (and its name) is based on the concept introduced by Ted Nelson in his Xanadu Project [Nelson 1980] to refer to the collection of text fragments that can be interconnected to each other and transcluded into new documents.

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

```
DatatypeProperty: hasNamespace Characteristics: FunctionalProperty
  Domain: MarkupItem Range: xsd:anyURI

Class: c:Collection
Class: c:Set SubClassOf: c:Collection
Class: c:Bag SubClassOf: c:Collection
Class: c:List SubClassOf: c:Bag
Class: c:Item
Class: c:ListItem SubClassOf: c:Item

ObjectProperty: c:element Domain: c:Collection

ObjectProperty: c:item SubPropertyOf: c:element
  Domain: c:Bag Range: c:Item

ObjectProperty: c:firstItem SubPropertyOf: c:item
  Domain: c:List

ObjectProperty: c:itemContent Characteristics: FunctionalProperty
  Domain: c:Item Range: not c:Item

ObjectProperty: c:nextItem Characteristics: FunctionalProperty
  Domain: c:ListItem Range: c:ListItem
```

A *markupitem* individual is a collection (`c:Set`, `c:Bag` or `c:List`, where the latter is a subclass of the second one and all of them are subclasses of `c:Collection`) of individuals belonging to the classes *MarkupItem* and *Range*. Through these collections it is possible to define a markup item as a set, a bag or a list of other markup items, using the properties *element* (for sets), *item* and *itemContent* (for bags and lists). Thus it becomes possible to define elements containing nested elements or text, or attributes containing values, as well as overlapping and complex structures. Note also that handling collections directly in OWL allows us to reason about content models for markup items, which would not be possible if we had used the corresponding constructs in RDF[13].

A *markupitem* might also have a name, specified in the functional property *hasGeneralIdentifier* (recalling the SGML term to refer to the name of elements [Goldfarb 1990]), and a namespace specified using the functional property *hasNamespace*. Note that we can have *anonymous* markup items – as it is possible in LMNL [Tennison & Piez 2002] and GODDAG [Sperberg-McQueen & Huitfeldt 2004] – by simply asserting that the item belongs to the class of all those markupitems that do not have a general identifier (i.e., `hasGeneralIdentifier exactly 0`).

4.2. Shell classes

The ghost classes discussed so far give us an abstract picture of the EARMARK framework. We need to specialize our model, defining a concrete description of our classes. These new *shell* subclasses apply specific restrictions to the *ghost* classes.

First of all, the class *Docuverse* is restricted to be either a *StringDocuverse* (the content is specified by a string) or an *URIDocuverse* (the actual content is located at the URI specified).

```
Class: StringDocuverse DisjointWith: URIDocuverse
  SubClassOf: Docuverse , hasContent some xsd:string
```

[13] http://hcklab.blogspot.com/2008/12/moving-towards-swan-collections.html

```
Class: URIDocuverse
  SubClassOf: Docuverse , hasContent some xsd:anyURI
```

Depending on particular scenarios or on the kind of docuverse we are dealing with – it may be plain-text, XML, LaTeX, a picture, etc. – we need to use different kinds of ranges. Therefore, the class *Range* has three different subclasses:

— *PointerRange* defines a range by counting characters. In that case, the value of the properties *begins* and *ends* must be a non-negative integer that identifies unambiguous positions in the character stream, remembering that the value *0* refers to the location immediately before the 1st character, the value *1* refers to the location after the 1st character and before the 2nd one, and so on. By using the *hasKey* OWL property, we also assert that two pointer ranges having equal docuverse, begin and end locations are the same range;
— *XPathRange* defines a range considering the whole docuverse or its particular context specifiable through an XPath expression [Berglund et al. 2007] as value of the property *hasXPathContext*. Note that, by using these ranges, we implicitly admit that the docuverse it refers to must be an XML structure. Moreover, the properties *begins* and *ends* have to be applied on the string value obtained by juxtaposing all the text nodes identified by the XPath. By using the *hasKey* OWL property, we also assert that two xpath ranges having equal docuverse, XPath context, begin and end locations are the same range;
— XPathPointerRange is an XPathRange in which the value of the properties begins and ends must be a non-negative integer that identifies unambiguous positions in the character stream as described for the class *PointerRange*.

```
Class: PointerRange HasKey: refersTo begins ends
  SubClassOf: Range ,
    begins some xsd:nonNegativeInteger and
    ends some xsd:nonNegativeInteger

Class: XPathRange SubClassOf: Range
  EquivalentTo: hasXPathContext some rdfs:Literal
  HasKey: refersTo begins ends hasXPathContext

Class: XPathPointerRange
  SubClassOf: XPathRange ,
    begins some xsd:nonNegativeInteger and
    ends some xsd:nonNegativeInteger

DatatypeProperty: hasXPathContext Characteristics: FunctionalProperty
  Domain: XPathRange Range: rdfs:Literal
```

MarkupItem is specialized in three disjointed sub-classes: *Element*, *Attribute* and *Comment*, that allow a more precise characterization of markup items.

```
Class: Element SubClassOf: MarkupItem
Class: Attribute SubClassOf: MarkupItem
Class: Comment SubClassOf: MarkupItem
DisjointedClasses: Element , Attribute , Comment
```

4.3. Range and markup item overlap

The presence of overlap in EARMAK is worth discussing more in detail. Different types of overlap exist – according to the subset of items involved – and different strategies are needed to detect them. In particular, there is a clear distinction between *overlapping ranges* and *overlapping markup items*.

By definition, *overlapping ranges* are two ranges that refer to the same docuverse and so that at least one of the locations of the first range is contained in the interval described by the locations of the second range (excluding its terminal points). *Totally overlapping ranges* have the locations of the first range completely contained in the interval of the second range or vice versa, while *partially overlapping ranges* have either exactly one location inside the interval and the other outside or identical terminal points in reversed roles.

Thus, if we consider the following excerpt:

```
Individual: r1 Types: PointerRange
  Facts: refersTo aDocuverse ,
    begins "0"^^xsd:nonNegativeInteger ,
    ends "7"^^xsd:nonNegativeInteger

Individual: r2 Types: PointerRange
  Facts: refersTo aDocuverse ,
    begins "4"^^xsd:nonNegativeInteger ,
    ends "9"^^xsd:nonNegativeInteger
```

we can infer, through a reasoner such as Pellet, that these two ranges overlap by using the following rules:

```
begins(x,b1) ^ ends(x,e1) ^ begins(y,b2) ^ ends(y,e2) ^
refersTo(x,d) ^ refersTo(y,d) ^ DifferentFrom(x,y) ^ P
  -> overlapWith(x,y)
```

where P is one of:

— lessThan(b1,e1) ^ greaterThan(b2,b1) ^ lessThan(b2,e1)
— lessThan(b1,e1) ^ greaterThan(e2,b1) ^ lessThan(e2,e1)
— lessThan(e1,b1) ^ greaterThan(b2,e1) ^ lessThan(b2,b1)
— lessThan(e1,b1) ^ greaterThan(e2,e1) ^ lessThan(e2,b1)

The case of *overlapping markup items* is slightly more complicated. We define that two markup items A and B **overlap** when at least one of the following sentences holds:

(1) **[overlap by range]** A contains a range that overlaps with another range contained by B;
(2) **[overlap by content hierarchy]** A and B contain at least a range in common;
(3) **[overlap by markup hierarchy]** A and B contain at least a markup item in common.

The three possible scenarios for such item overlap are summarized in Fig. 2[14].

The EARMARK ontology, in fact, is completed by another ontology[15] that models all overlapping scenarios, either for ranges or markup items, and includes rules for inferring overlaps automatically, through a reasoner.

4.4. EARMARK as a standoff notation

If we ignore for a moment the semantic implications of using EARMARK and concentrate on its syntactical aspects only, it is easy to observe that EARMARK is nothing but yet another standoff notation, where the markup specifications point to, rather than contain, the relevant substructure and text fragments.

[14]The EARMARK documents describing these three overlapping scenarios and all the other ones presented in the following sections are available at http://www.essepuntato.it/2011/jasist/examples.
[15]http://www.essepuntato.it/2011/05/overlapping

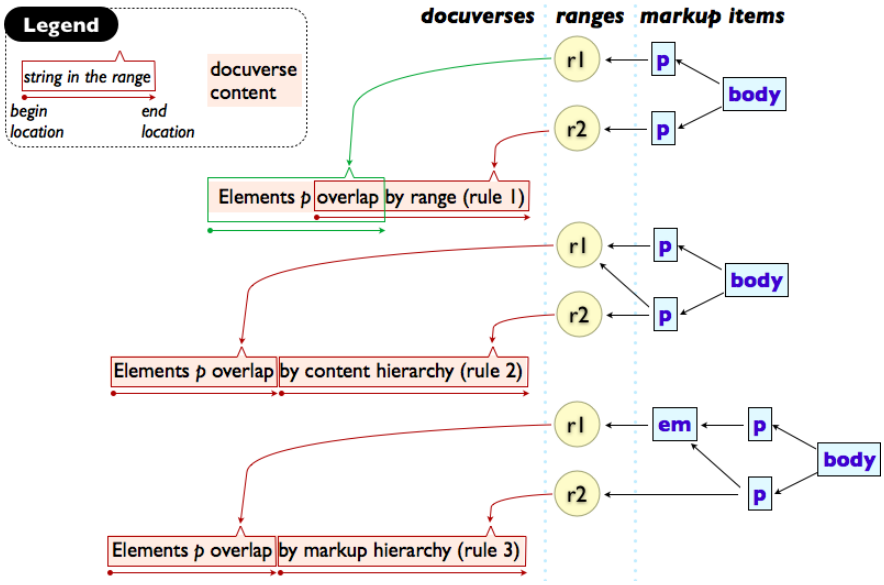

Fig. 2. Three EARMARK examples of overlapping between elements *p*.

Standoff notations, also known in literature as out-of-line notations [TEI Consortium 2005], are hardly new, but never really caught on for a number of reasons, most having to do with their perceived fragility under the circumstances of desynchronized modification to the text. In [Georg et al. 2010] and [Baski 2010] we can find a pair of recent and substantially complete analysis of their merits and demerits. In particular, according to , "standoff annotation has [...] quite a few disadvantages:

(1) very difficult to read for humans
(2) the information, although included, is difficult to access using generic methods
(3) limited software support as standard parsing or editing software cannot be employed
(4) standard document grammars can only be used for the level which contains both markup and textual data
(5) new layers require a separate interpretation
(6) layers, although separate, often depend on each other"[16].

And yet, although EARMARK *is* in practice a standoff notation, it provides a number of workarounds to most of the above-mentioned issues.

Firstly, since EARMARK is based on OWL and can be linearized in any of the large number of OWL caricaturisation syntaxes, it follows that 1) readability, 2) access and 3) software support for it are exactly those existing for well-known, widespread and important W3C standards such as RDF and OWL. Being able to employ common RDF and OWL tools such as Jena and SPARQL for EARMARK documents was in fact a major motivation for it.

Issue 4 should be examined beyond the mere validation against document grammars and towards a general evaluation of the level of compliancy of the markup to some formally specified expectations. EARMARK documents, while being subject to no document grammar in the stricter XML sense, allow the specification of any number

[16]In order to individually address the issues, we edited the original bullets into a numbered list.

of constraints, expressed either directly in OWL, or in SWRL [Horrocks et al. 2004] or even in SPARQL, that trigger or generate validity evaluations. In [Di Iorio et al. 2011] we tried to show that a large number of requirements, from hierarchical well-formedness in the XML sense, to validation requirements in terms of XML DTDs, to adherence to design patterns, can be expressed satisfactorily using these technologies.

Item 5 regards the difficulty of standoff notations to provide inter-layer analysis on XML structures: separate interpretation of markup layers is easy, but identification and validation of overlapping situations is more complex: standoff markup is mainly composed of pointers to content, and does not have any direct way to determine overlap locations without some kind of pointer arithmetics to compute them. Validation of contexts allowing overlaps as describable using rabbit-duck grammars [Sperberg-McQueen 2006] is also not trivial. In this regard EARMARK provides yet again a solution that does not require special tools: although OWL does not allow direct pointer arithmetics, SWRL on the contrary does, as shown in Section 4.3 where we described a batch of (SWRL-implementable) rules that do in fact determine overlapping locations on EARMARK documents with good efficiency.

Finally, issue 6 refers to the fact that evolution of separate markup annotation layers need to take place synchronously, lest one of them become misaligned with the new state of the document. This is, in summary, the *fragility of pointers*, which can be considered the fundamental weakness of standoff, as well as of any notation that has markup separate from its content: if a modification occurs to the underlying (probably text-based) source, all standoff pointers that could not be updated at the same time of the change become outdated and possibly wrong. All standoff notations fall prey of this weakness, and there is no way to completely get rid of it.

What is possible is to identify exactly what are the conditions under which such weakness acts, and see if there is a way to reduce the mere frequency of such events. In fact, in order for a standoff pointer to become outdated, several conditions must take place at the same time:

— the standoff notation must be used as a storage format, rather than just as a processing format;
— the source document must make sense even without the additional standoff markup (i.e., the standoff notation contains no information that is necessary for at least some types of document modifications);
— the source document must be editable (and, in fact, must be edited) on its own;
— the standoff pointers must rely on positions that change when the source is edited (e.g., character-based locations);
— editing must be done in contexts and with tools that cannot or do not update the standoff pointers;
— there must be no computable way to determine the modifications of the document (e.g. via a *diff* between the old and the new version).

Of course, no standoff notation can rule out that these conditions occur on their documents. But it is worth pointing out that **all six** of them must occur, for standoff pointers to become outdated. EARMARK is not safe from these occurrences, either, but, at least for the use cases here described, one or more of these conditions simply do not apply: EARMARK is mostly used as a processing format, with no need to save it on disk (conversion from the source formats, e.g. MS Word, is described in Section 6 and does not require special storage), the data format described is either in a very specific format (such as MS Word or ODT) that in fact already does handle internally its data changes and requires the overlapping data exactly for this purpose, or is in fact the result of a diff action on successive versions of a document (as in the case of the wiki pages). Finally, EARMARK allows references to relatively stable fragment ids

of the documents (by using XPath ranges without specifying explicitly begin and end locations), rather than the extremely fragile character locations, further reducing the chances of outdated pointers.

For this reason, without being able to completely rule out the possibility of standoff pointers to go wrong, we tend to consider it as a significantly little risk, at least for the use case here described.

4.5. Using OWL vs. RDF for standoff notations

EARMARK is strongly based on OWL 2 DL [W3C OWL Working Group 2009] to express multiple markup layers with possible overlapping ranges over the same content. OWL 2 DL is not the only possible choice for expressing standoff notations via Semantic Web technologies. In fact, RDF is another valid and effective model for dealing with the same issue, as shown in [Tummarello et al. 2005] by means of the open-source API *RDF Textual Encoding Framework* (*RDFTef*). This API was created to demonstrate a plausible way for handling overlapping markup within documents and identifying textual content of a document as a set of independent RDF resources that can be linked mutually and with other parent resources.

Beside giving the possibility to define multiple structural markup hierarchies over the same text content, the use of RDF as the language for encoding markup allows to specify semantic data on textual content as well. But the real main advantage in using RDF is the possibility of using particular built-in resources appositely defined in the RDF syntax specification [Beckett 2004] for describing and dealing with different kinds of containers, either ordered (rdf:Seq) or unordered (rdf:Bag). Thus, RDF resources can be used to represent every printable element in the text – words, punctuation, characters, typographical symbols, and so on – while RDF containers can be used to combine such fragments and containers as well.

Although RDF is not sufficient to define a formal vocabulary for structural markup – does a given resource represent an element, an attribute, a comment or a text node? In which way is a resource of a certain type related to others? – the specification of an RDFS [Brickley & Guha 2004] or of an OWL layer can successfully address this issues. Hybrid solutions obtained by mixing different models, even when they are built one upon another, may seem elegant but not necessarily the best choice. In fact, there exist well-known interoperability limits between OWL 2 DL and RDF that prevent the correct use of Semantic Web tools and technologies. In particular:

— any markup document made using RDF containers (e.g. to describe what markup items contain and in which order) and OWL ontologies (e.g. to define classes of markup entities and their semantics) results in an set of axioms that end up outside of OWL DL and well within OWL Full, which limits the applicability of the most frequently used Semantic Web tools, that are usually built upon the (computationally-tractable) description logic underlying OWL 2 DL;

— the individual analysis of each language may be not applicable when we have to check particular properties that lay between RDF and OWL layers. For example, verifying the validity of a markup document against a particular schema, which is one of the most common activities with markup, needs to be made to work with both markup item structures (that would be defined in RDF) and logical constraints about classes of markup items (e.g., elements only, attributes only, the element “p”, all the element of a particular namespace, etc., all of them definable in OWL).

Being able to express everything we need directly in OWL addresses both issues quite straightforwardly. The well known absence of containers and sequences in OWL can be overcome by modeling classes in specific ways using specific design patterns such as [Ciccarese et al. 2008] and [Drummond et al. 2006].

5. USING EARMARK

There are multiple applications for the EARMARK approach. The most interesting for this paper is its capability of dealing with overlapping structures in an elegant and straightforward manner. Under EARMARK such structures do not need to be specified through complex workarounds as with XML, but they are explicit and can be easily described and accessed. Sophisticated searches and content manipulations become very simple when using this ontological model.

The goal of this section is to demonstrate the soundness and applicability of EARMARK by discussing how the use-cases presented in Section 3 are addressed. Notice that throughout the section we investigate multiple EARMARK data structures and documents, focussing on the feasibility and potentiality of such an ontological representation.

5.1. Looking for authorial changes in Office Documents

The discussion in Section 3.1 showed that both ODT (OpenOffice format) and OOXML (Microsoft Word format) use complex data structures to store overlaps generated by change-tracking functionalities. These structures make it very difficult to search and manipulate the content when using XML languages and tools. Even very simple edits generate a rather tangled set of overlapping elements.

Let us recall the example mentioned in Section 3.1, where the user "John Smith" splits a single paragraph into two. The ODT representation is:

```
<text:tracked-changes>
  <text:changed-region text:id="S1">
    <text:insertion><office:change-info>
        <dc:creator>John Smith</dc:creator>
        <dc:date>2009-10-27T18:45:00</dc:date>
    </office:change-info></text:insertion>
  </text:changed-region>
  [... other changes ...]
</text:tracked-changes>
[... content ...]
<text:p>The beginning and
  <text:change-start text:change-id="S1"/></text:p>
<text:p> also<text:change-end text:change-id="S1"/> the end.</text:p>
```

The OOXML representation , as shown in Section 3.1, is even more complex. In fact these formats exploit in large scale (tangled) fragmentation (OOXML), or milestones and stand-off markup (ODT) to deal with overlaps.

EARMARK, on the other hand, stores overlapping data in a direct and streamlined manner that does not require tools to rebuild information from the twists of a tree-based XML structure. The information is already available and expressed through consistent RDF and OWL statements. Fig. 3 graphically shows the corresponding EARMARK document.

The original paragraph content and the new string "also" are now encoded as two *docuverses* over which the ranges *r1*, *r2* and *r3* are defined. The original paragraph is then composed of the (content of) ranges *r1* and *r2,* while the paragraphs resulting after the (text and carriage return) insertion now comprise respectively range *r1* and ranges *r2*, *r3*. Metadata about the author and the modification date are encoded as further RDF statements.

```
Individual: doc1 Types: StringDocuverse
  Facts: "The beginning and the end"
Individual: doc2 Types: StringDocuverse Facts: " also"
```

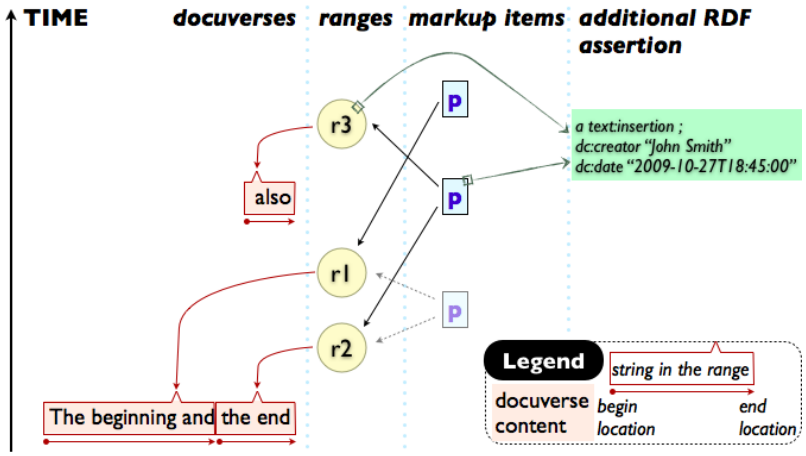

Fig. 3. Encoding in EARMARK the ODT change-tracking example.

```
Individual: r1 Types: PointerRange
  Facts: refersTo doc1 , begin "0"^^xsd:nonNegativeInteger ,
    end "17"^^xsd:nonNegativeInteger
Individual: r2 Types: PointerRange
  Facts: refersTo doc1 , begin "17"^^xsd:nonNegativeInteger ,
    end "25"^^xsd:nonNegativeInteger
Individual: r3 Types: PointerRange , insJS
  Facts: refersTo doc2 , begin "0"^^xsd:nonNegativeInteger ,
    end "5"^^xsd:nonNegativeInteger

Individual: p-b Types: Element Facts: firstItem p-b-i1
Individual: p-b-i1 Facts: c:itemContent r1 , nextItem p-b-i2
Individual: p-b-i2 Facts: c:itemContent r2

Individual: p-m Types: Element , insJS
  Facts: firstItem p-m-i1
Individual: p-m-i1 Facts: c:itemContent r3 , nextItem p-m-i2
Individual: p-m-i2 Facts: c:itemContent r2
Individual: insJS Types: Insertion
  Facts: dc:creator "John Smith" , dc:date "2009-10-27T18:45:00"

Individual: p-t Types: Element Facts: firstItem p-t-i
Individual: p-t-i Facts: c:itemContent r1
```

The advantages of streamlining overlaps becomes apparent if we consider tasks a little beyond the mere display. For instance, the query for "the textual content of all paragraphs inserted by John Smith" ends up rather entangled if we used XPath on the ODT structure. The process for finding that textual content needs to browse the current version of the document, look for all the *text:change-start*/*text:change-end* pairs that refer to an insertion made by John Smith involving the creation of a new paragraph (i.e., *text:change-start* is in a first paragraph while its pair, *text:change- end*, is in the following one), that are either currently present in the document body or hidden behind a subsequent deletion made by someone else. Once identified the paragraphs, we need to retrieve the content that originally was contained there, i.e., the text fragments that still are within those boundaries or that may have been deleted in

subsequent versions. The following XPath represent an implementation of the above process:

```
for $cr in (//text:changed-region) , $date in ($cr/text:insertion//(
    @office:chg-date-time | dc:date)) return $cr[.//text:insertion
    [(.//@office:chg-author = 'John Smith' and count($cr//text:p) = 2)
     or (.//dc:creator = 'John Smith' and (//text:change-start[@text:
    change-id = $cr/@text:id]/following::text:p intersect //text:
    change-end[@text:change-id = $cr/@text:id]/(ancestor::text:p)))]]/
    root()//((text:change-start[@text:change-id = $cr/@text:id]/(
    following::text:p//((text()|(for $tc in (text:change) return //
    text:changed-region[@text:id = $tc/@text:change-id and not(text:
    insertion//(@office:chg-date-time | dc:date) > $date)]//text:p
    [1]//text())) except ((for $tc in (text:change) return $tc[count
    (//text:changed-region[@text:id = $tc/@text:change-id and not(text
    :insertion//(@office:chg-date-time | dc:date) > $date)]//text:p) =
     2]/following::text()) union (//text:changed-region/text:deletion
    [.//dc:date <= $date]/text:p//text())))))) | (text:change[@text:
    change-id = $cr/@text:id]/(following::text()[ancestor::text:p] | (
    for $tc in (following::text:change) return //text:changed-region[
    @text:id = $tc/@text:change-id and not(text:insertion//(@office:
    chg-date-time | dc:date) > $date)]//text:p[1]//text())) except ((
    for $tc in (following::text:change) return $tc[count(//text:
    changed-region[@text:id = $tc/@text:change-id and not(text:
    insertion//(@office:chg-date-time | dc:date) > $date)]//text:p) =
    2]/following::text()) union (//text:changed-region/text:deletion
    [.//dc:date <= $date]/text:p//text())))) | $cr//text:p[2]//text())
    except (//(text:change|text:change-end)[@text:change-id = $cr/
    @text:id]//following::text:p[not((text:change-end|text:change-
    start|text:change)[1]/self::text:change-end)]/(. | following-
    sibling::element())//((text() | (for $tc in (text:change) return //
    text:changed-region[@text:id = $tc/@text:change-id]//text())))
```

The XML structure of a MS Word file, using segmentation rather than milestones, does simplify a bit the query, but still presents some radical complexities. The process starts by choosing all those *w:p* elements that were inserted by John Smith, as well as all their previous and contiguous *w:p* elements that were deleted before or inserted after the first ones. In OOXML, each sequence of contiguous *w:p* elements represents implicitly one paragraph. Therefore, we can now take all the text fragments contained in each *w:p* sequence that were inserted before or deleted after the paragraph defined by the sequence itself. The following is the resulting XPath for an OOXML document .

```
for $p in (//w:p[w:pPr//w:ins/@w:author = 'John Smith']) return for
    $date in ($p/w:pPr//w:ins/@w:date) return ($p|($p/preceding-
    sibling::w:p[w:pPr//w:del[@w:date <= $date]|w:pPr//w:ins[@w:date >
     $date]] except $p/preceding-sibling::element()[not(self::w:p) or
    empty(w:pPr//w:del[@w:date <= $date]|w:pPr//w:ins[@w:date > $date
    ])]/(.|preceding-sibling::element()))//(w:t|w:delText)[empty(
    ancestor::w:ins[@w:date > $date]|ancestor::w:del[@w:date <= $date
    ])]
```

The complexity of both XPath queries is due to the intrinsic complexity of the data structure the query has to work on. Although the interface of OpenOffice or MS Word may provide tools to directly deal with these queries using specific strategies on the internal data structures, applications working directly on the XML structure have very little help in disentangling the mess of the data formats.

On the other hand, since EARMARK documents are actually OWL files it is possible to access and query them with plain Semantic Web tools. Powerful searches can be then performed without using niche-specific tools or complex and long XPath expressions but simply with mainstream technologies such as SPARQL [Garlik & Seaborne 2010].

The corresponding SPARQL query for ("the textual content of all paragraphs inserted by John Smith") can therefore be written as follows:

```
SELECT ?p ?r WHERE {?r a :Range .
  ?p :hasGeneralIdentifier "p" ; :hasNamespace "http://..."
    ; a [ a :Insertion ; dc:creator "John Smith" ]
    ; :item/:itemContent ?r . }
```

But EARMARK is useful for even more than querying: EARMARK also decreases the costs, in terms of efforts and lines of code, for manipulating documents.

Let us consider the task of generating an intermediate version (i.e., neither the first nor the last one of a version chain), from a document that includes change-tracking information about the whole document history.

The process of rebuilding these versions by working on the XML structure without specific APIs is complex and inefficient at the same time. For example a basic XSLT that returns an XML document defining the desired version requires at least to:

— define templates for all the elements actively involved in the change tracking – e.g., for ODT, *text:changed-region*, *text:change-start*, *text:change-end* and *text:change*, and similarly for OOXML– in order to understand, by looking at their creation date, whether they must be considered or ignored when building the requested version. In particular, we must exclude insertions following and deletions preceding the version we are building;

— define templates for paragraphs, in order to handle cases where the paragraph is the result of an insertion or a deletion of other paragraphs, in order to identify whether it should be considered for the result and, in such case, finding out its real text content and remembering that, in the following versions, such content may have spread out among other paragraphs;

— define templates for handling insertions/deletions for structures such as images, sections, lists, and tables;

— define an identity template for the other elements, in order to visit the entire document.

Even the most basic and incomplete implementation of such XSLT requires hundreds of lines of complex and convoluted code and a large number of *ad hoc* decisions based on the specificities of whether we start from ODT or OOXML. Notice also that a Java-based implementation (or in any other procedural language) of the same process would be equally or even more complex.

The same result can be achieved on EARMARK documents with a few lines of Java code:

```
public EARMARKDocument getVersion(
  EARMARKDocument d, String creator, String date) {

  private final boolean RECURSIVELY = true;
  EARMARKDocument version = null;
  String query =
  "PREFIX : <http://www.essepuntato.it/2008/12/earmark#>" +
  "PREFIX c: <http://swan.mindinformatics.org/ontologies/1.2/
     collections/>" +
  "PREFIX dc: <http://purl.org/dc/elements/1.1/>" +
  "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
```

```
  "SELECT ?root" +
   "WHERE {" +
   "?root a :Element ; dc:creator \"" + creator +
   "\"; dc:date \"" + date + "\"." +
  "FILTER NOT EXISTS { ?mi a :MarkupItem ;" +
   "c:item/c:itemContent ?root . } }";

  EARMARKElement theRoot = d.findNode(query) ;
  version = new EARMARKDocument(
    URI.create("http://www.example.com/version"));
  version.copyNode(theRoot, RECURSIVELY);
  return version;
}
```

This approach uses the EARMARK Java API[17] and a single SPARQL query, runnable on any SPARQL 1.1 processor such as Jena, to identify the root node of the subtree of the version that is associated with the specified date and creator. Then, it performs a simple recursive deep-first visit in order to clone all the nodes in the tree and to combine them in the output EARMARK document.

This method heavily uses Semantic Web technologies on the structures provided by EARMARK whose characteristics are always explicit and clear. In fact, since *all* versions coexist within the EARMARK document and each version can be encoded *explicitly* as a tree within the overall graph, this operation is straightforward and fast.

5.2. Improving Semantic Annotations

EARMARK can also be exploited to improve semantic annotations. As pointed out in Section 3.2, in fact, there are strong limitations in the same process of annotating web documents with semantic structures that overlap the structural ones. The same example - of *vcards* that cannot be created on the top of tables organized *per rows* – will be used in this section.

We solve this by converting the Web document with annotations into an EARMARK document allowing both semantic and structural annotations to coexist. Through EARMARK, we can explicitly express both markup structures and *vcard* assertions. Fig. 4 shows how the *vcard* example can be modeled (once again we show a graphical representation for the sake of clarity).

The textual content of the original table cells is now encoded in two different docuverses, one for the header (with roles) and one for the body (with names of committee members). Ranges *r1*, *r2*, *..., r8* are then created to distinguish each role and name. Two independent and coexisting hierarchies are then built on top of the same set of ranges: the HTML table that includes one cell for each range (in blue) and the *Vcards* about each person (in green) that include only the relevant ranges and overlap the previous one. Notice also that the *Vcards* are defined in such a way that does not interfere with the structural features of the table. The full linearization in OWL of this example can be found at http://www.essepuntato.it/2011/jasist/examples.

5.3. Improving Wiki Content Reversions

EARMARK can be used to improve wiki reversion mechanisms and overcome the limitations discussed in Section 3.3: the automatic filtering and merging of contributions from multiple versions of the same page is now still a *manual* process, but it can be fully automatized if the overlapping structures buried in the *whole* history of the page become explicit.

[17] http://earmark.sourceforge.net

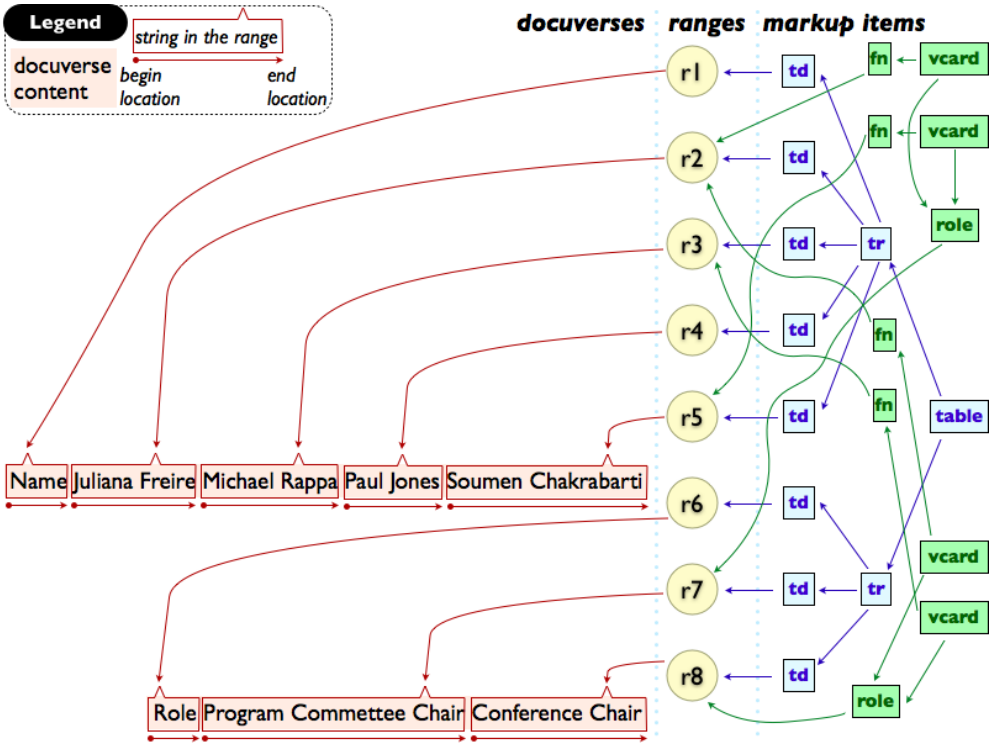

Fig. 4. The abstract model of the EARMARK document solving the microformats issue.

The role of EARMARK is to make those structures explicit and available for more sophisticated content manipulation. In order to understand to what extent EARMARK structures can be derived from wikis and how they can be exploited by the final users, we use as our example the wiki platform MediaWiki[18], i.e., the wiki engine of Wikipedia.

MediaWiki offers sophisticated functionalities for creating *diffs* of wiki content. Users can compare any two revisions in the page history and highlight changes in a friendly interface that shows modifications with a word-level granularity. *Diff* pages contain metadata about each compared version (when the version was created, who was the author or which IP address an anonymous author was connected from, etc.) and a two-column table showing the changes side-by-side. Changes are detected *a posteriori* by comparing two arbitrary versions, not even requiring them to be temporally contiguous.

The output of the MediaWiki diff engine has regularities that can be exploited to automatically build the overlapping structures of the *diff* and to express them in EARMARK. Let us consider a fictitious example summarized in Table I, where an initial text is revised three times by different authors.

To display the differences between V1 ad V2, Mediawiki creates a page whose HTML code is as follows[19]:

```
<table class="diff"><tbody>
  <tr valign="top">
```

[18] http://www.mediawiki.org

[19] For the sake of clarity we removed all markup irrelevant to our discussion.

Table I. All the versions of a wiki page modified by different authors.

Version	V1	V2	V3	V4
Author	151.61.3.122	Angelo Di Iorio	Silvio Peroni	Fabio Vitali
Content	Bob was farming carrots and tomatoes	Bob was farming carrots, tomatoes **and beans**	Bob was farming carrots, tomatoes and **green** beans. **They were all tasteful .**	Bob was farming carrots, tomatoes and green beans. *[new paragraph]* They were all tasteful.

```
    <td class="diff-otitle">
      <div id="mw-diff-otitle1">
        <a href="{...}oldid=413">Revision as of 15:46, 8 November
            2009</a></div>
      <div id="mw-diff-otitle2"><a>151.61.3.122</a></div></td>
    <td class="diff-ntitle">
      <div id="mw-diff-ntitle1">
        <a href="{...}oldid=414">Revision as of 15:47, 8 November
            2009</a></div>
      <div id="mw-diff-ntitle2">Angelo Di Iorio</div></td></tr>
  <tr>
    <td class="diff-marker">-</td>
    <td class="diff-deletedline">
      <div>Bob was farming carrots <del class="diffchange diffchange-
          inline">and </del>tomatoes.</div></td>
    <td class="diff-marker">+</td>
    <td class="diff-addedline">
      <div>Bob was farming carrots
        <ins class="diffchange diffchange-inline">
          , </ins> tomatoes
        <ins class="diffchange diffchange-inline">
          and beans</ins>.</div></td></tr>
</tbody></table>
```

This is an HTML table of two rows, the first showing metadata (date and author of the modification) and the second the actual modifications. The first cell of the second row contains all the unmodified text and a *del* element for each inline fragment that was deleted. The second cell contains all the unmodified text and an *ins* element for each inline fragment that was inserted. Thus, these cells share *exactly* the same unmodified part(s) of the two compared versions.

When the structure itself is modified, rather than merely the text, the source code of the MediaWiki *diff* is slightly different. Thus the diff between V3 and V4 (which splits a paragraph in two) is as follows:

```
<tr>
  <td class="diff-marker">-</td>
  <td class="diff-deletedline">
    <div>Bob was farming carrots, tomatoes and green beans. They were
        all tasteful.</div></td>
  <td class="diff-marker">+</td>
  <td class="diff-addedline">
    <div>Bob was farming carrots, tomatoes and green beans. </
       div></td></tr>
<tr>
  <td colspan="2"> </td><td class="diff-marker">+</td>
  <td class="diff-addedline"><div> </div></td></tr>
<tr>
```

```
<td colspan="2"> </td><td class="diff-marker">+</td>
<td class="diff-addedline">
  <div>They were all tasteful.</div></td></tr>
```

The *diff* output is not complete nor sophisticated, and of course it is a completely different task to re-plan such algorithm (but for a first idea of natural changes in *diffing* XML documents, see [Di Iorio et al. 2009]). Thus, limitations of that algorithm are inevitably shared by any EARMARK representation. Yet, this output is sufficiently rich to allow us to extract the overlapping information we need. For instance, the insertion of a non-breakable-space or a carriage-return generate rows according to specific rules that can be easily detected to capture the actual change by the author.

Fig. 5 shows the above example rebuilt in EARMARK. All versions are encoded in the same document by creating overlapping assertions over the docuverses. Metadata and RDF statements are layered on top of those assertions and create a rich knowledge-base about the history of the documents and, in particular, about the history of each fragment.

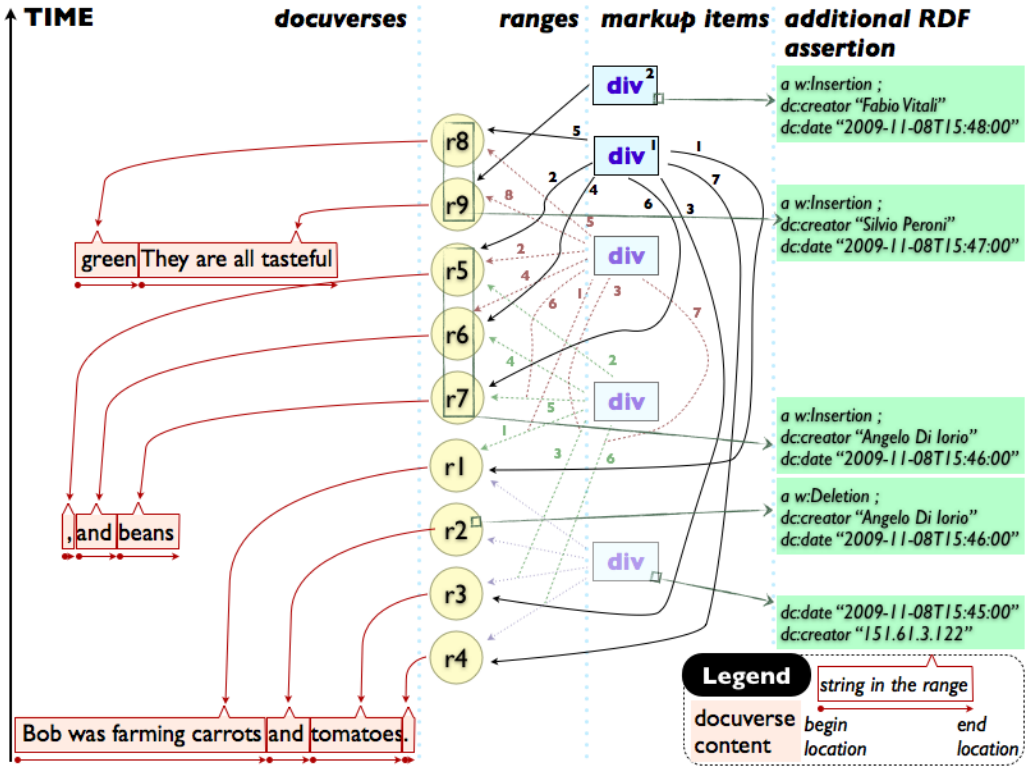

Fig. 5. The wiki sample versions encoded in a single EARMARK document.

Due to the complexity of the example we labeled arrows with numbers indicating the position of each range within each markup item. Consider for instance version *V4*: it is composed of two DIV elements, the first one containing the concatenation of “Bob was farming carrots” + “,” + “tomatoes” + “and” + “green” + “beans” + “.”, and the second one contains the string “They are all tasteful”.

Implementing a wiki content filtering mechanism on top of such a structure is rather simple. For instance, the removal of all the contributions of "Angelo Di Iorio", that leaves untouched all the content written (previously and subsequently) by "Silvio Peroni" and "Fabio Vitali", can be performed straightforwardly. Three steps are enough to apply such an intermediate content reversion:

(1) the identification of the fragments written by "Angelo Di Iorio", which is a straightforward SPARQL query on the embedded statements;
(2) the creation of a new version where references to those fragments are removed and references to fragments no longer in the document are correctly fixed;
(3) the translation of that document into an actual MediaWiki page through the serialization process described in [Peroni & Vitali 2009].

Of course, an automatic process may generate ambiguities or even errors in the resulting content (some parts may become dangling, wrong or unclear after removing text fragments elsewhere); grammar discrepancies might also be generated by the same approach. Linguistic and semantic problems, however, become a problem once the technical issues of managing independent yet successive edits are solved. What is important is that all the information about overlaps and dependencies among fragments are available in EARMARK and can easily be searched, filtered and manipulated. Besides, foreseeing a manual intervention for checking and polishing automatically-filtered content is perfectly in line with the wiki philosophy, so that the wiki community itself can use the reversion tools wisely to revise the content and adjust any intervening minor nuisances or imperfections. Such checks would still be far simpler and faster than the manual process of partially reverting versions as we have today.

6. GENERATING EARMARK FROM EXISTING DOCUMENTS: THE ROCCO APPROACH

Since we do not expect documents to be natively written in EARMARK or manually created by users, we need a way to extract EARMARK data structures from existing XML-based resources, which is trivial when the XML is simple and clearly hierarchical, and slightly more complex when the XML contains workarounds to force an intrinsically overlapping situation into a single hierarchy.

We designed a reliable process to transform XML files into EARMARK documents that fully capture overlapping structures even when the overlaps is hidden in one the many well-known workarounds. This approach takes as input an XML file and produces the corresponding EARMARK document in five steps: *Read*, *Overhaul*, *Convert*, *Classify* and *Organize* (hence the name ROCCO).

Since ROCCO is not the main topic of this paper we discuss the issue of converting XML into EARMARK very briefly, explaining how each step works.The ROCCO algorithm performs five steps, described in the following sections.

6.1. Read and Overhaul

The first two steps consist of loading the XML source file and, if needed, adding information useful for further processing. In EARMARK there is a clear distinction between the textual content of a document and the structures built on top of it: the content is stored as plain text – within *docuverses* – and all structures are externalized and expressed through OWL and RDF assertions.

While OpenOffice stores all overlapping structures in the main document file, some other editors (e.g. MS Word) store overlaps in many different ways, even in a separate file. The *overhaul* step extracts such data and adds them to the main content document by exploiting format-specific procedures, implemented via XSLT in most cases.

6.2. Convert

The subsequent step consists of converting the XML source file into an early EARMARK document that expresses *exactly* the same information and hierarchies. No interpretation or disentanglement of workarounds is performed at this step.

Since the input is XML, this translation can be performed directly via a generic XSLT stylesheet. It basically consists of a recursive algorithm that parses the source file and generates the corresponding instances in the EARMARK ontology. Such a translation is straightforward and not difficult.

6.3. Classify

The "Classify" step extends the EARMARK document built so far with information about the workarounds used to encode overlaps. That information will be exploited in the subsequent steps in order to make those overlaps explicit.

The basic idea is to exploit OWL reasoners to detect workarounds in an early EARMARK document D by:

— defining an ontology O that models all the workarounds used by applications, such as milestones, stand-off markup, etc.; these workarounds are specific of the data format used in the source document;
— specifying the EARMARK document D as an ABox for the ontology O;
— defining SWRL rules that capture the role of each element in D and check relationships between elements;
— running an OWL reasoner, such as Pellet, on D+O in order to create new OWL instances and properties that identify which workarounds are present.

The actual detection of workarounds is delegated to an external reasoner. Refining detection strategies and even adding new strategies for new formats can all be done via OWL and SPARQL. Indeed, tricky issues need to be addressed – mostly depending on the idiosyncrasies of the original formats – but no procedural code is required.

6.4. Organize

The final step consists of building yet another EARMARK document that expresses the overlaps and metadata *in an explicit way*, based on the information collected by the previous steps.

This phase consists of mapping operations from the native format into the EARMARK structure. Such conversion relies on the identification of metadata in order to classify the operations and to externalize relevant metadata in separate RDF statements.

7. EVALUATING EARMARK

One of the most frequent criticisms when proposing a different approach to solving a well-known problem in ICT is that the new solution may simplify the difficulties of the specific problem, but brings with it hidden costs in terms of size of the data structure, computation efforts or conversions restrictions that compensate the advantages. In our case, one of the anonymous reviewers of our paper [Di Iorio et al. 2009] wondered whether a difference in file size could weigh in on the convenience of adopting EARMARK as opposed to working with the original files.

As such, a discussion of cost functions of EARMARK versus other formats is in order. Yet, a systematic discussion of the relative costs (e.g., in byte size) of some original XML-based data structures versus their EARMARK equivalent is an open-ended undertaking that heavily depends on the original XML data structure and the specific features present in the document, and is badly defined anyway: while XML is a lin-

earization format immediately expressible in actual bytes, OWL (or, more precisely, RDF, the language in which OWL ontologies are expressed) is an abstract structure that allows a large number of linearization formats (including XML itself) with corresponding huge differences in the final byte counts.

For these reasons, in order to provide at least an initial test of meaningful concepts, we selected two XML-based data formats (OOXML and ODT), and specifically a set of documents where overlapping tricks were present (i.e., where change-tracking was active). And to bypass the size discussion, we decided to test not byte-lengths (which are not meaningful and easily skewed, e.g., by reducing the string length of the element names or of the class names), but the number of nodes for XML documents and of triples for OWL documents. This comparison is again not particularly appropriate (triples are naturally numerous in OWL ontologies, and it is customary to deal with hundreds of thousands and even millions of assertions in Semantic Web applications), but closer to meaningfulness than mere byte count.

Our comparison was carried on a small set of documents in ODT and OOXML that included change-tracking information. As discussed in the previous sections, change-tracking facilities generate rather complex overlaps even for basic operations on small text fragments, which in turn are expressed as a potentially huge number of standoff and milestone markup within the XML hierarchy. The same documents were individually converted into EARMARK. We then charted how simple edits under change-tracking affect the number of nodes in XML formats and of statements in OWL files[20].

We created seven different versions, named after the Seven Dwarfs for recognizability, by applying very common edits (the insertion of few words, the deletion of some sentences, the split of a paragraph, and so on) on a small document, creating multiple overlaps. Fig. 6 shows the results of our comparison.

The overall trend is interesting and comforting: while in simple documents with no overlap the node count of XML is lower than the assertion count of EARMARK triples, the presence of overlaps makes EARMARK and XML formats comparable. The growth of EARMARK statements is in fact very close to the growth of XML nodes when the number of overlaps increases. EARMARK is even more efficient than XML for more complex documents.

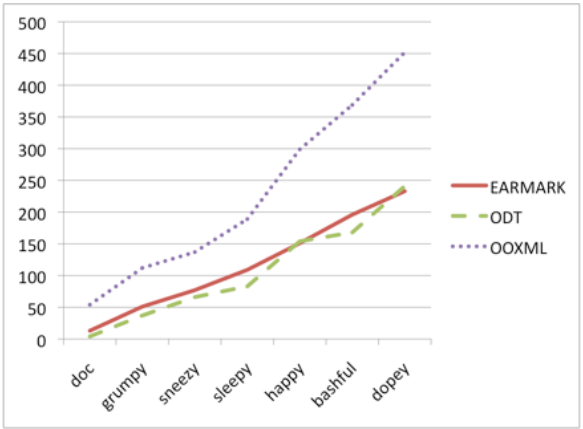

Fig. 6. A graph summarizing the results of the first experiment.

[20]The full details about each version and each format are also available at http://www.essepuntato.it/2011/jasist/discussion.

The measure for each format was done by counting only those nodes and statements instrumental to encode content and (overlapping) structures: we did not take into account neither the presentational information for ODT and OOXML (each file, for instance, includes a very long list of style definitions that are not relevant for the purposes of our analysis) nor namespace declarations (OOXML files, for instance, lists all relevant namespaces for the Office toolkit) nor ignorable white-spaces (that are only added to indent content and improve readability).

Interestingly, EARMARK and ODT show a very similar increase in size, while OOXML is much more verbose and grows faster. The content of the first version, for instance, is encoded using 4 nodes in ODT, 13 statements in EARMARK and 54 nodes in OOXML; the last one contains 241 ODT nodes, 233 EARMARK statements and 452 OOXML nodes. To return to our original enquiry, anyway, it is clear that the weight of EARMARK documents is very good compared to the other ones.

It is also worthy of note the regularity in the growth of EARMARK statements. Regardless of the actual modifications applied to the document, in fact, EARMARK adds about 40 statements for each edit. Both OOXML and ODT, on the contrary, show a more irregular "pace". The reason is that EARMARK externalizes *all* assertions, so that *all* modifications (either to leaf-nodes or to intermediate nodes in the original XML) are "flattened" onto the docuverses and do not depend on the complexity of the structure within which the edit took place.

Fig. 7 shows the results of a similar comparison on a different set of documents and edits. We collected seven versions named after the weekdays and created by seven different authors when editing a very simple document. The overall trend does not change and shows that EARMARK and ODT have again a comparable behavior, far better than OOXML.

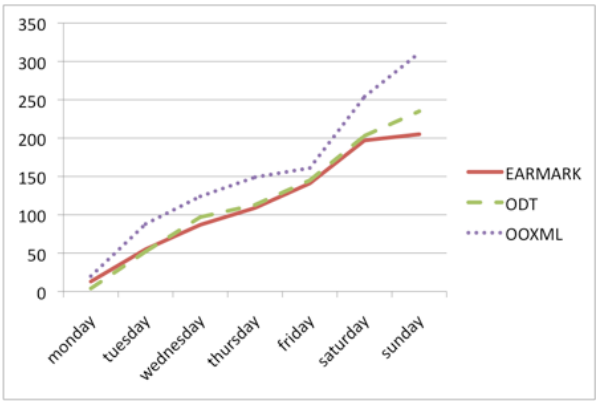

Fig. 7. A graph summarizing the results of the second experiment.

In conclusion, although preliminary, this study shows clear trends of a very conservative behavior of EARMARK with respect to document size.

8. CONCLUSIONS

Overlaps, far from being an obscure requirement for sophisticated functionalities of arcane markup languages, is a very frequent undertaking even in major data formats and in rather frequent situations. Yet, since the XML language does not allow them, consciously or not designers of data formats have adopted a huge and entangled array of tricks, special cases and workarounds that, although solving the actual problem

of storing overlapping structures, open new and complicated ones when approaching even basic chores on documents containing them, such as queries.

The EARMARK approach reduces drastically the efforts needed to perform such chores on overlapping structures, since it does not allow the corresponding multiple trees to actually entangle and complicate the job. EARMARK is radically different from both special markup metalanguages that allow overlaps and the introduction of workarounds within the traditional tree-oriented XML language, because at the same time treats multiple trees over the same content as first class citizens of the language, yet uses well-known and standard W3C technologies and languages to perform all tasks. EARMARK documents, at the end, are OWL ontologies. Thus any Semantic Web technology – such as SPARQL – can be used *straightforwardly* to perform operations on their content.

Improving queries is not the only application of EARMARK. Validation is another interesting field we are investigating, The same ontological framework, in fact, can be used to prove properties concerning a document such as validity against a schema, compliance to co-constraint specifications or adherence to structural patterns. Moreover, inspired by [Marcoux & Rizkallah 2009] in which authors describe an approach for defining natural-language semantics for XML-based languages, we are also developing an ontology-based approach for encoding *markup semantics* – i.e., the formal definition of meanings of markup elements, besides the syntactical structure of a markup document – within EARMARK documents.

REFERENCES

Adida, B., Birbeck, M., McCarron, S., & Pemberton, S. (2008). RDFa in XHTML: Syntax and processing. W3C Recommendation 14 October 2008, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/rdfa-syntax/

Allsopp, J. (2007). Microformats: Empowering Your Markup for Web 2.0. New York, NY, USA: Friends of ED Press.

Baski, P. (2010). Why TEI stand-off annotation doesn't quite work: and why you might want to use it nevertheless. In Proceedings of Balisage: The Markup Conference 2010. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://www.balisage.net/Proceedings/vol5/html/Banski01/BalisageVol5-Banski01.html

Beckett, D. (2004). RDF/XML Syntax Specification (Revised). W3C Recommendation 10 February 2004, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

Berglund, A., Boag, S., Chamberlin, D., Fernndez, M. F., Kay, M., Robie, J., & Siméon, J. (2007). XML Path Language (XPath) 2.0. W3C Recommendation 23 January 2007, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/xpath20/

Brickley, D., & Guha, R.V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/rdf-schema/

Ciccarese, P., Wu, E., Kinoshita, J., Wong, G., Ocana, M., Ruttenberg, A., & Clark, T. (2008). The SWAN Biomedical Discourse Ontology. Journal of Biomedical Informatics, 41 (5), 739-751.

DeRose, S. (2004). Markup Overlap: A Review and a Horse. In Proceedings of the Extreme Markup Languages 2004. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://conferences.idealliance.org/extreme/html/2004/DeRose01/EML2004DeRose01.html

Di Iorio, A., Marchetti, C., Schirinzi, M., & Vitali, F. (2009). Natural and Multi-layered Approach to Detect Changes in Tree-based Textual Documents. In J. Cordeiro, & J. Filipe (Eds.), Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS 2009) (pp. 90-101). Heidelberg, Germany: Springer.

Di Iorio, A., Peroni, S., & Vitali, F. (2009). Towards markup support for full GODDAGs and beyond: the EARMARK approach. In Proceedings of Balisage: The Markup Conference 2009. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://balisage.net/Proceedings/vol3/html/Peroni01/BalisageVol3-Peroni01.html

Di Iorio, A., Peroni, S., & Vitali, F. (2010). Handling markup overlaps using OWL. In P. Cimiano, & H. S. Pinto (Eds.), Proceedings of the 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2010) (pp. 391-400). Heidelberg, Germany: Springer.

Di Iorio, A., Peroni, S., & Vitali, F. (2011). Using Semantic Web technologies for analysis and validation of structural markup. Submitted for publication in International Journal of Web Engineering and Technologies.

Drummond, N., Rector, A., Stevens, R., Moulton, G., Horridge, M., Wang, H. H., & Seidenberg, J. (2006). Putting OWL in Order: Patterns for Sequences in OWL. In B. C. Grau, P. Hitzler, C. Shankey, & Evan Wallace (Eds.), Proceedings of the Workshop on OWL: Experiences and Directions (OWLED 2006). Athens, Georgia, USA.

Durand, D. G. (1994, October). Palimpsest, a Data Model for Revision Control. Paper presented at the Workshop on Collaborative Editing Systems, within the Computer Supported Cooperative Work Conference (CSCW94), Chapel Hill, North Carolina, USA.

Durand, D. G. (2008). Palimpsest: Change-Oriented Concurrency Control for the Support of Collaborative Applications. Charleston, SC, USA: CreateSpace.

Garlik, S. H., & Seaborne, A. (2010). SPARQL 1.1 Query Language. W3C Working Draft 14 October 2010, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/sparql11-query/

Georg, R., Schonefeld, O., Trippel, T., & Witt, A. (2010). Sustainability of Linguistic Resources Revisited. In Proceedings of the International Symposium on XML for the Long Haul: Issues in the Long-term Preservation of XML. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://www.balisage.net/Proceedings/vol6/html/Witt01/BalisageVol6-Witt01.html

Goldfarb, C. F. (1990). The SGML Handbook.New York, NY, USA: Oxford University Press.

Horridge, M., & Patel-Schneider, P. (2009). OWL 2 Web Ontology Language: Manchester Syntax. W3C Working Group Note 27 October 2009, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/owl2-manchester-syntax/

Horrocks, I., Patel-Schneider, P. F., Boley, H. Tabet, S., Grosof, B., & Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/Submission/SWRL/

Huitfeldt, C., & Sperberg-McQueen, C. M. (2003). TexMECS: An experimental markup meta-language for complex documents [Working paper]. Retrieved May 12, 2011, http://decentius.aksis.uib.no/mlcd/2003/Papers/texmecs.html

JTC1/SC34 WG 4. (2008). ISO/IEC 29500-1:2008 – Information technology – Document description and processing languages – Office Open XML File Formats – Part 1: Fundamentals and Markup Language Reference. Geneva, Switzerland: International Organization for Standardization.

JTC1/SC34 WG 6. (2006). ISO/IEC 26300:2006 – Information technology – Open Document Format for Office Applications (OpenDocument) v1.0. Geneva, Switzerland: International Organization for Standardization.

Marcoux, Y., & Rizkallah, E. (2009). Intertextual semantics: A semantics for information design. Journal of the American Society for Information Science and Technology, 60 (9), 1895-1906.

Marinelli, P., Vitali, F., & Zacchiroli, S. (2008). Towards the unification of formats for overlapping markup. New Review of Hypermedia and Multimedia, 14 (1), 57-94.

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. Sausalito, CA, USA: Mindful Press.

Peroni, S., & Vitali, F. (2009). Annotations with EARMARK for arbitrary, overlapping and out-of order markup. In U. M. Borghoff, & B. Chidlovskii (Eds.), Proceedings of the 2009 ACM Symposium on Document Engineering (DocEng 2009) (pp. 171-180).New York, NY, USA: ACM.

Portier, P., & Calabretto, S. (2009). Methodology for the construction of multi-structured documents. In Proceedings of Balisage: The Markup Conference 2009. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://balisage.net/Proceedings/vol3/html/Portier01/BalisageVol3-Portier01.html

Riggs, K.R. (2002). XML and Free Text. Journal of the American Society for Information Science and Technology, 53 (6), 526-528.

Salembier, P., & Benitez, A. B. (2007). Structure description tools. Journal of the American Society for Information Science and Technology, 58 (9), 1329-1337.

Schmidt, D. (2009). Merging Multi-Version Texts: a Generic Solution to the Overlap Problem. In Proceedings of Balisage: The Markup Conference 2009. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://balisage.net/Proceedings/vol3/html/Schmidt01/BalisageVol3-Schmidt01.html

Schmidt, D., & Colomb, R. (2009). A data structure for representing multi-version texts online. In International Journal of Human-Computer Studies, 67 (6), 497-514.

Schonefeld, O., & Witt, A. (2006). Towards validation of concurrent markup. In Proceedings of the Extreme Markup Languages 2006. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://conferences.idealliance.org/extreme/html/2006/Schonefeld01/EML2006Schonefeld01.html

Sperberg-McQueen, C. M. (2006). Rabbit/duck grammars: a validation method for overlapping structures. In Proceedings of Extreme Markup Languages Conference 2006. Rockville, MD, USA: Mulberry Technologies, Inc. Retrieved May 12, 2011, http://conferences.idealliance.org/extreme/html/2006/SperbergMcQueen01/EML2006SperbergMcQueen01.html

Sperberg-McQueen, C. M., & Huitfeldt, C. (2004). GODDAG: A Data Structure for Overlapping Hierarchies. In P. R. King, & E. V. Munson (Eds.), Proceeding of the 5th International Workshop on the Principles of Digital Document Processing (PODDP 2000) (pp. 139-160). Heidelberg, Germany: Springer.

TEI Consortium (2005). TEI P5: Guidelines for Electronic Text Encoding and Interchange. TEI Consortium. Retrieved May 12, 2011, http://www.tei-c.org/Guidelines/P5

Tennison, J., & Piez, W. (2002, August). The Layered Markup and Annotation Language (LMNL). Presented at the Extreme Markup Languages Conference 2002, Montreal, Canada.

Tummarello, G., Morbidoni, C., & Pierazzo, E. (2005). Toward Textual Encoding Based on RDF. In Milena Dobreva, & Jan Engelen (Eds.), Proceedings of the 9th ICCC International Conference on Electronic Publishing (ELPUB2005). Leuven, Belgium: Peeters Publishing Leuven.

W3C OWL Working Group (2009). OWL 2 Web Ontology Language Document Overview. W3C Recommendation 27 October 2009, World Wide Web Consortium. Retrieved May 12, 2011, http://www.w3.org/TR/owl2-overview/