Overapproaches in documents: a definitive classification (in OWL, 2!)

Silvio Peroni  
Department of Computer Science and Engineering, University of Bologna, Bologna, Italy  
<silvio.peroni@unibo.it>

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

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

*Balisage: The Markup Conference 2014*  
August 5 - 8, 2014

Copyright © 2014 by the authors.

---

**How to cite this paper**


---

**Abstract**

Several different types of overlap exist and different strategies are needed to detect them. In particular, there is a clear difference between ranges of text that overlap and markup items that overlap (that is, elements and attributes), and how these types of overlapping affect dominance and containment relations of nodes is of some relevance, too. In order to provide a complete definition and description of these overlapping patterns, we introduce the *EARMARK Overlapping Ontology* (*EOO*), i.e., an OWL 2 DL ontology that extends *EARMARK* (an OWL-based markup meta-language compliant with extended GODDAGs) to define properties describing dominance and containment relations as well as a complete characterisation of the different kinds of overlap that can happen to nodes. In addition, we also present some inference rules for the automatic retrieval (by means of a reasoner) of all the overlapping instances in a given input markup document.

---

**Table of Contents**

Introduction  
Overlapping markup: a summary for the absent and the distracted  
Forcing overlaps in plain XML  
Non-XML syntaxes for overlaps  
EARMARK  
Characterizing overlaps by way of an ontology  
Properties of overlapping
Introduction

At the Balisage 2009 Conference we presented for the first time a new approach to overlapping markup called EARMARK, or "Extremely Annotational RDF Markup". It provided a point of view over something that is quintessentially Balisagean, overlapping markup, by using a number of suspicious techniques for this community, such as standoff markup, RDF, OWL, reasoners.

In brief, an EARMARK document is a collection of RDF statements about fragments of a text (a plain text or even an XML document), that describe the fragments' characteristics and features regardless of whether the fragments contain, are disjoint, or overlap with each other. Each fragment is associated to a formal concept called Range, which can (but does not have to) be associated to one or more Markup Items, which in turn may, or may not, refer to each other in some form. Since these annotations (and the objects they represent) are never embedded in the text, there are no implicit properties to consider, in particular no properties indirectly provided by the fragments' position in the text, relative to each other, according to document's order, etc. Thus in an EARMARK document a property exists (e.g., A contains B) if and only if it has been explicitly stated in the ontology, and not just because they happen to refer to the same text fragment.\footnote{[1]}

In doing so, EARMARK manages not only to be the only overlapping approach that fully expresses and makes use of unrestricted GODDAG, the formal model introduced in [34] by Sperberg-McQueen and Huitfeldt, but actually corresponds to a non-trivial extension of it, e-GODDAG, that supports repetitions of the same node in different contexts, in addition to self-overlap, discontinuous overlap, anonymous nodes, decoupling of containment and dominance, etc.

The conjunction of stand-off as a referencing approach, and RDF as the assertion syntax, allows EARMARK to bypass completely the usual dichotomy of embedded markup, that of either hiding overlapping situations inside a traditional, hierarchical XML markup, tricky but conservatively transparent with respect to the most common XML tools and services, or inventing a completely new syntax and having to deal with the lack of the usual validation tools, transformation tools, storage systems, etc. On the contrary, an EARMARK document is just a collection of RDF statements, and plain and usual RDF and OWL tools can be used to manage it: inference engines, rule-based systems, query languages, and triple stores work transparently with overlapping data, and any existing and future tool for RDF and OWL will be available for use transparently when managing EARMARK documents, too \footnote{[2]}.

But in our 2009 paper, we actually and quite conventienly avoided to discuss a rather relevant issue: EARMARK, then, did not really allow to describe overlapping markup situations, but rather it allowed to describe traditional markup situations that could refer to overlapping content. Since each markup statement is independent of the others, it can refer to partially or totally overlapping ranges and children with no need (and no possibility) to determine that such overlap has actually happened.

But if you really want to be able to determine whether and where overlapping has happened in an EARMARK document, you need a few more tools, that luckily can and have been realized using
standard and well-known RDF and OWL tools. In this paper, we present the EARMARK Overlapping Ontology (EOO), an ontology that uses OWL 2 and SWRL to provide a complete characterization of overlapping situations in EARMARK documents, allowing queries and representations that discovers and manages explicitly (instead of simply allowing and ignoring) all overlaps in the markup. This characterization takes the form of definitions of the overlapping patterns of the basic EARMARK ontology, and is therefore a definitive classification \[3\] of the overlapping approaches of EARMARK documents.

Of particular relevance for the EOO ontology is being able to distinguish between different aspects/manifestations of the overlap phenomenon, such as, for example:

- **total vs. partial overlap:** we talk about total overlap to refer to those situations where one item is completely contained by the other, without breaking the rules imposed by the tree hierarchy, while we use partial overlap to indicate cases where no hierarchy can exist, as only part of the content is shared by the items;

- **dominance vs. containment:** we use these terms to distinguish and discern between cases where there really is a hierarchical relation between items that overlap (dominance), from situations in which items just happen to refer to the same content (containment);

- **range overlap vs. markup overlap:** we can also distinguish between items that overlap by sharing the same textual content (range overlap) from situations in which items overlap by insisting either partially or totally on the same elements (markup overlap).

The paper is organized as follows: in the next section, we present a brief overview of the topics, results and languages that have been proposed to handle overlaps in markup documents. In section “EARMARK”, EARMARK is (re-)introduced with its main classes and concepts. In section “Characterizing overlaps by way of an ontology”, the EARMARK Overlapping Ontology is presented and described, as well as how it can be used to identify and characterize the overlapping situations of an EARMARK document. The following section contains our conclusions and hints at future works.

**Overlapping markup: a summary for the absent and the distracted**

When marking up text documents it might be necessary to represent features that do not fit into the tree structure conveyed by an XML document. In fact, there are many situations in which authors may need to annotate the same piece of text with different markup descriptors (e.g. when a page spans from the middle of one paragraph to the middle of another, or when speeches span multiple verses, etc.): in such cases, the markup descriptors sometimes nest correctly into a single tree-hierarchy, sometimes not. In general, this issue may arises whenever an author wants to maintain two or more views of a document (e.g. metrical, syntactical, layout, etc.), and consequently multiple and incompatible hierarchies insists on the same textual content. This problem is referred to in the literature as the overlapping problem.

After a first period in which the deficiencies of markup languages that concerns the overlapping problem were overlooked \[3\] [4] or even suppressed \[9\] [5], the digital humanities community started to put an increasing effort in trying to define and develop solution to this issue. The essence of the problem can be summarized as follows: “overlap can be presented by graphs that are very like trees, but in which nodes may have multiple parents. Overlap is multiple parentage” \[34\]. While trying to represent non-hierarchical structures using a markup language whose model is a tree, such as SGML or XML, authors run into different manifestations of the problem, referred to using different
terminology in the literature:

- **classic overlap**: this is the most common case of overlap, that consists in two markup elements with different general identifiers that share a part of their textual content. This situation occurs whenever two document fragments that need to be annotated with different markup descriptors overlap each other. Typically this scenarios arises when authors want to merge multiple concurrent hierarchies over the same document, e.g. phontetical, grammatical and typographical structures.

- **self overlap**: the term “self” overlap is used to refer to that situations in which two components of the same structure, and with the same name, overlap each other. A typical example is a document that should be commented by two different reviewers: whenever they need to annotate two text fragments that overlap, two elements of the same structure (the comment structure) and with the same name overlap each other.

- **out-of-order elements**: there are also cases in which the content of an element is a reordering of information present elsewhere in the document. For example, sometimes it would be useful to define elements whose content is not a continuous text region (we refer to such cases as discontinuous elements), or to express more complex features, such as out-of-order or repeated uses of the same text fragment, etc. The general approach used by embedded markup languages to deal with such cases is to use a technique called virtual elements: the information needed to convey such features is encoded by using an ad-hoc mechanism, such as a linking system by means of elements' attributes. The term “virtual” is used because these elements are not explicitly present in the document, but their presence may be inferred by an external application from the specific encoding mechanism supplied [35].

- **containment/dominance decoupling**: Most of the solutions to the problem of overlapping markup implicitly leave unwanted relations between the concurrent hierarchies. The best known is the identity between dominance and containment. Dominance is a relation between document parts where one is said to dominate another if it is one of its ancestors in the document structure. Containment is rather a look at two document parts from the point of view of which slices of the actual character content of the document they enclose; a document part contains another one if it encloses all the character content of that other part. Tree-based markup languages such as XML have the (implicit) property that containment implies dominance, but in general this is neither desirable nor correct. Consequently, most of the approaches to the overlapping problem that forces multiple hierarchies (i.e. graphs) into a single tree structure reflects this limitation. Moreover, most of the complexity in the process of managing these document is due to this reason, since it requires an external and often conceptual effort to understand, interpret and correctly manage dominance as separated from containment.

Since the document model of XML is inherently a tree, there is no simple way to cover such complex situations when handling multiple hierarchies. In order to overcome these limitations, many different solutions have been proposed. In general, we can identify two different approaches to the problem.

The first consists in devising techniques to encode the information about overlapping situations by using specific XML features (e.g. empty elements to specify the boundary of overlapping elements, attributes to link elements that don't nest properly, etc.) or technologies (e.g. XPath [4], XQuery [5], etc.). The second approach is to abandon XML altogether and with it the benefits of its tree-based data model, and devise a new formalism and notation based on a more general and expressive abstract structure, such as a directed graph.

**Forcing overlaps in plain XML**

Documents in XML-based formats have the advantage that any existing application, tool and technology can be used to process them, at the cost of a post-parsing processing in order to
reconstruct and correctly handle the not tree-based structures coerced using these conventions. The main drawback of this approach is that the overlapping situations encoded in XML-based formats are neither easy to read, write and understand by humans without the help of specific tools, since these techniques considerably increase the complexity of the resulting XML document. Moreover, the process of forcing multiple hierarchies (i.e. a graph) into a single tree structure used by most of these techniques often introduces unwanted dominance relations between elements belonging to different hierarchies, and these situations need a further (and usually manual) effort in order to be identified, properly interpreted and managed.

The universe of the XML-based techniques to manage overlapping situations is quite ample. We summarize the four most used mechanisms [28]:

- **TEI-style milestones**: this approach is to represent a vocabulary as primary by using a standard XML structure, and to use pairs of empty elements to mark the boundaries of elements that belong to secondary vocabularies. In order to make explicit the relation between corresponding opening and closing empty tags, a co-indexing mechanism may be implemented by means of special linking attributes[35][6];

- **fragmentation**: is another technique that envisions/prescribes to break the elements belonging to secondary hierarchies in as many smaller fragments (also called partial elements) as needed to nest properly into the primary hierarchy. Also in this case overlapping elements are linked using special attributes (e.g. id-idref or next-previous pairs).

- **stand-off markup**: the key idea is to represent hierarchical and possibly incompatible structures separately from their actual content. Infact, the real content is present elsewhere, for example within the same document or in separate ones, and included by means of links implemented through a pointer mechanism such as XPointer [10]. In this way, it is possible to represent multiple conflicting structures as stratifications of different layers, at the cost of a overhead to manage and keep up-to-date the referenced content not directly embedded within these structures.

- **twin documents**: overlapping hierarchies may also be encoded by using multiple documents that share the same textual content, but each one denoting its own tree structure.

In order to describe the expressiveness power of these techniques, in Table I we summarises their capability to manage the complex overlapping features introduced in the previous section.

<table>
<thead>
<tr>
<th>XML techniques / complex document features</th>
<th>Classic overlap</th>
<th>Self overlap</th>
<th>out-of-order elements</th>
<th>Containment/dominance decoupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milestones</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Stand-off markup</td>
<td>Yes*[17]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Twin documents</td>
<td>Yes*[17]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

In order to overcome the limitations of XML, many different solutions have been proposed:

- **CONCUR** [18] is an SGML option that allows multiple DTDs for the same content: all these structures live in the same document, and it is up to the parser to either consider the structure of
only one DTD, or parse them simultaneously but keeping separate track of what elements are open in each. The main advantage of this technique is that documents are quite legible and maintainable, but there are many drawbacks: for example, it is not possible to constrain relationships across DTDs, it is not possible to express self-overlap situations, and there is little software support for this technique;

- **JITT (Just In Time Trees):** another syntax very close to XML have been proposed [14] [15]. The basic idea is similar to CONCUR in that it requires the parser to filter and take in consideration only some tags: multiple overlapping hierarchies may coexist into documents, but only those which the filter selects are returned to the application as real start or end tags. JITT’s main contribution is that a document need not be well-formed until the moment it is being processed, at the cost of a very small change to an XML parser. Unfortunately, JITT’s does not provide a way to correlate and validate across structures, and it is not possible to express cases of self-overlap.

- **MuLaX:** another document syntax similar to SGML CONCUR for XML called MuLaX has been developed [19] together with a constraint based validation language [29] [37]. Each overlapping hierarchy represents a layer identified by an ID prefixing each tag name, and multiple layers may coexist into one MuLaX document. An external software can parse a MuLaX document and project each layer into well formed XML documents. Standard XML tools can only be used on these separate XML projections. A drawback of this technique is that these documents can get very complex when dealing with a large number of annotation layers: for example, updates are difficult since working on MuLaX documents requires frequent projections into XML projections. Moreover, the project is still at the state of experimental markup languages, lacking the support of tools and technologies as that available for XML-based solutions.

- **Multi-colored trees:** another extension of the XML model that is able to represent overlapping structures are the Multi-colored trees [24]. The basic idea is to associate a color to each concurrent tree, and to allow each node to have multiple colors. Navigation inside the multicolored nodes is possible by using an Xpath [4] extension that implements a color selector, and an extension of XQuery [5] has also been proposed for the creation of nodes.

**Non-XML syntaxes for overlaps**

An alternative approach to overcome the limitations of tree-based meta-languages in representing complex documents is to use alternative and more expressive data models, such as graphs. The more general is the model (acyclic vs. cyclic graphs, ordered vs. unordered graphs, etc.), the more expressive is the meta-language in terms of overlapping features that can be conveniently managed, at the cost of an increased computational complexity. Moreover, since this abstract model may be represented with different concrete syntaxes (embedded markup languages, stand-off annotations, etc.), the chosen linearisation format may place limits in terms of expressiveness, support provided by standard technologies and related tools, etc. A summary of the most eminent solutions is presented below:

- **GODDAG and TexMECS:** Sperber-McQueen and Huittfeld proposed to manage overlapping hierarchies using a directed acyclic graph structure with no transitive arcs named GODDAG (General Ordered Descendant Directed Acyclic Graph)[34]. Arcs denotes containment relationships, and multi-parentage is allowed, thus making it possible to represent overlapping situations. Several kinds of GODDAG have been defined in order to explore their expressive power and their mutual relation: generalized, restricted and clean in [34], normalized and colored in [23], node-ordered (noDAG) in [26], child-arc-ordered in [27]. The authors of GODDAG also developed a markup meta-language named TexMECS [22] as the natural linearisation format for the GODDAG structure. As XML, TexMECS is an embedded meta-markup language where elements are delimited by start and end tags, but it also allows to
represent graph structures by allowing tags to not nest properly. TEXMECS supports complex document features, such as self overlap (using a co-indexing scheme) and discontinuous, virtual and unordered elements (using special attributes and elements' delimiters). Since TEXMECS documents are not isomorphic to XML documents, the standard XML tools cannot be used and, as far as we know, no query mechanisms have been developed.

- **LMNL**: the Layered Markup and Annotation Language [36] defines a specific syntax based on layered ranges which can overlap each other. A LMNL document is a set of layers containing either a sequence of Unicode characters (text layer) or a sequence of ranges. A layer can be based on a single other layer, but can also be the base of several other layers. LMNL is able to capture classic and self overlap cases and virtual elements (via a pointers' mechanism), but since a range spans over continuous sequences of characters, there is no way to represent discontinuous text fragments and element with mixed content (i.e. characters and other ranges). Despite the main contribution of LMNL is a data model, at least three syntaxes have been proposed: two are XML-based (ECLIX [7] and CLIX [8], both based on the milestone technique), and a non-XML syntax known as the LMNL syntax. XSLT stylesheets have been developed to deal with the XML representation of a LMNL document.

**EARMARK**

The Extremely Annotational RDF Markup, or EARMARK [12], is an OWL 2 DL ontology[7] that defines document meta-markup. It is an ontologically precise definition of markup that instantiates the markup of a text document as an independent OWL document outside of the text strings it annotates, and through appropriate OWL and SWRL characterisations it can define structures such as trees or graphs (in particular, extended GODDAGs [11]) and can be used to generate validity constraints (including co-constraints) [13], to make explicit the semantics of markup [30], to annotate text or other markup documents [1], to keep track of changes in markup [31], and as interchange format to enable conversions between different kinds of XML vocabularies embedding overlap [2]. The whole ontological description of EARMARK is summarised in the Graffoo diagram[8] [16] shown in Figure 1.

**Figure 1:**
The core classes of our model describe three disjoint base concepts: docuverses, ranges and markup items.

The textual content of an EARMARK document is conceptually separated from its annotations, and is referred to through the earmark:Docuverse class. The individuals of this class represent the objects of discourse, i.e. all the containers of text from an EARMARK document. Any individual of the earmark:Docuverse class – commonly called a docuverse (lowercase to distinguish it from the class) – specifies its actual content through the property earmark:hasContent. There exist two different kinds of docuverses, those that specify all its content in form of a string (defined through the class earmark:StringDocuverse) and those that refer to a document containing the string to be marked up (defined through the class earmark:URI_docuverse).

We define the class earmark:Range for any text lying between two locations of a docuverse. A range, i.e., an individual of the class earmark:Range, is defined by a starting and an ending location (any literal) of a specific docuverse through the functional properties earmark:begins, earmark:ends and earmark:refersTo respectively. There exist two main types of ranges: those (i.e., earmark:PointerRange) that refer to text lying between two non-negative integer locations that identify precise positions within a docuverse, and those (defined through the class earmark:XPathPointerRange) that refer to any text, obtained from a particular XPath context (specified through the property earmark:hasXPathContext) starting from a docuverse content, lying between two non-negative integer locations that identify precise positions.

The class earmark:MarkupItem is the superclass defining artefacts to be interpreted as markup such as elements (i.e., the class earmark:Element), attributes (i.e., the class earmark:Attribute) and comments (i.e., the class earmark:Comment). A markup item individual is a collection\(^9\) (co:Set, co:Bag and co:List, where the latter is a subclass of the second one and all of them are subclasses of co:Collection) of individuals belonging to the classes earmark:MarkupItem and earmark:Range. Through these collections it is possible:

- to define a markup item as a set of other markup items and ranges by using the property co:element;
- to define a markup item as a bag of items (defined by individuals belonging to the class co:Item), each of them containing a markup item or a range, by using the properties c:item and co:itemContent respectively;
- to define a markup item as a list of items (defined by individuals belonging to the class co:ListItem), each of them containing a markup item or a range, in which we can also specify a particular order among the items themselves by using the property co:nextItem.

A markup item might also have a name, specified in the functional property earmark:hasGeneralIdentifier\(^10\), and a namespace specified using the functional property earmark:hasNamespace.

In order to understand how EARMARK is used to describe markup hierarchies, let us consider the markup structures shown in Figure 2.
First of all, we define the whole textual content of the document – i.e., the first three lines of the *Paradise Lost* by John Milton – by creating an instance of the class *earmark:StringDocuverse*[^11]:

```
@prefix : <http://www.essepuntato.it/2014/balisage/example/> .
:doc a earmark:StringDocuverse ;
  earmark:hasContent
    "Of Mans First Disobedience, and the Fruit
    Of that Forbidden Tree, whose mortal tast
    Brought Death into the World" .
```

Then, we can define all the six different ranges (as individuals of *earmark:PointerRange*) that are introduced in the figure, i.e.:

```
# The string 'Of Mans First Disobedience, and the Fruit'
:r1 a earmark:PointerRange ;
  earmark:refersTo :doc ;
  earmark:begins "0"^^xsd:nonNegativeInteger ;
  earmark:end "41"^^xsd:nonNegativeInteger .

# The string 'the Fruit Of that Forbidden Tree,'
:r2 a earmark:PointerRange ;
  earmark:refersTo :doc ;
  earmark:begins "32"^^xsd:nonNegativeInteger ;
  earmark:end "65"^^xsd:nonNegativeInteger .

# The string 'Of that Forbidden Tree,'
:r3 a earmark:PointerRange ;
  earmark:refersTo :doc ;
  earmark:begins "42"^^xsd:nonNegativeInteger ;
  earmark:end "65"^^xsd:nonNegativeInteger .
```

Finally, we can built the three markup hierarchies shown in upon these ranges, as shown in the following excerpt:
Characterizing overlaps by way of an ontology

Different types of overlap exist – according to the subset of EARMARK nodes involved (i.e., ranges or markup items) – and different strategies are needed to detect them. In particular, there is a clear distinction between overlapping ranges and overlapping markup items, and in the ways these overlapping scenarios affect the dominance and containment relations between nodes – as shown in figure Figure 2, that will be used to illustrate the different kinds of overlapping scenarios.

In this section, we introduce the EARMARK Overlapping Ontology (EOO)\(^{[12]}\), which is an OWL 2 DL ontology [MotikOWL2] that extends the EARMARK Ontology by adding support for overlapping scenarios and for inferences relative to them. In particular, in the following subsections we describe how the ontology models all possible overlapping scenarios between nodes by means of description logic formulas\(^{[13]}\) and SWRL rules [21] (if needed)\(^{[14]}\). A summary of the taxonomy of possible overlapping scenarios is provided in figure Figure 3.

Figure 3:

![Diagram showing the taxonomy of all the different kinds of overlapping scenarios that are introduced in this paper.](image)

**Properties of overlapping**

The most important property in EOO is the generic property, \textit{eoo:overlapsWith}, that describes when
an EARMARK node overlaps with another EARMARK node of the same type. This means that markup items can overlap only with other markup items, and ranges can overlap only with ranges. In addition, this property is symmetric (i.e., if A overlaps with B, then B overlaps with A) and irreflexive (i.e., if A overlaps with B, then A is different from B\(^1\)). This property is defined formally as follows:

```plaintext
# Declaration as an object property
eeo:overlapsWith ⊆ Top

# Domain
∃eeo:overlapsWith. ⊆
 (earmark:Range ∩ eeo:overlapsWith.Range) ∪
 (earmark:MarkupItem ∩ eeo:overlapsWith.MarkupItem)

# Range
 ⊆ eeo:overlapsWith.( (earmark:Range ∩ eeo:overlapsWith.Range) ∪
 (earmark:MarkupItem ∩ eeo:overlapsWith.MarkupItem))

# Symmetry
eeo:overlapsWith = eeo:overlapsWith-

# Irreflexivity
 ⊆ ¬∃eeo:overlapsWith.Self
```

All the properties presented in the following sections are sub-properties of the generic relation `eeo:overlapsWith`.

### Overlapping of ranges

By definition, overlapping ranges (i.e., linked through the symmetric property `eeo:overlapsWithRange`) are two ranges of the same type that refer to the same docuverse and so that at least one of the end points of the first range is contained in the interval described by the locations of the second range (end-points excluded). The property `eeo:overlapsWithRange` is defined as follows:

```plaintext
# Sub-property declaration
eeo:overlapsWithRange ⊆ eeo:overlapsWith

# Domain
∃eeo:overlapsWithRange. ⊆ earmark:Range

# Range
 ⊆ eeo:overlapsWithRange.earmark:Range

# Symmetry
eeo:overlapsWithRange = eeo:overlapsWithRange-
```

Specifically, **totally overlapping ranges** (defined through the property `eeo:overlapsTotallyWithRange`) have the locations of the first range completely contained in the interval of the second range or vice versa, i.e., the range is fully contained inside the second range. For instance, in the example in Figure 2, the range “the Fruit Of that Forbidden Tree” overlaps totally with the range “Of that Forbidden Tree”.

On the other hand, partially overlapping ranges (defined through the property `eeo:overlapsPartiallyWithRange`) have exactly one location inside the interval and the other outside. For instance, considering the example in Figure 2, the range “Of Mans First Disobedience, and the Fruit” overlaps partially with “the Fruit Of that Forbidden Tree”. These two properties are disjoint, meaning that two ranges cannot overlap totally and partially between them. Additionally, this
property also handles the situation in which the two locations are completely identical, but the end points have reversed roles (i.e., the starting point of the first range is the ending point of the second one, and vice versa). They are formally defined as follows:

# Sub-property declarations
`eoo:overlapsTotallyWithRange ⊆ eoo:overlapsWithRange`
`eoo:overlapsPartiallyWithRange ⊆ eoo:overlapsWithRange`

# Disjointness
`eoo:overlapsTotallyWithRange n eoo:overlapsPartiallylyWithRange ⊆ ⊥`

# Symmetry
`eoo:overlapsTotallyWithRange ≡ eoo:overlapsTotallyWithRange`
`eoo:overlapsPartiallyWithRange ≡ eoo:overlapsPartiallyWithRange`

The following SWRL rules allows us to catch the constraints of this kind of overlap by inferring the overlapping relation between the two different kinds of (concrete) ranges, i.e., `earmark:PointerRange` and `earmark:XPathPointerRange`:\[16\]:

# Overlaps partially with range
`RANGE_IDENTIFICATION` ^
`earmark:refersTo(?x,?d) ^ earmark:refersTo(?y,?d) ^
earmark:begins(?x,?b1) ^ earmark:begins(?y,?b2) ^
earmark:ends(?x,?e1) ^ earmark:ends(?y,?e2)` ^

(?b1 < ?b2 < ?e1 < ?e2) or (?b1 < ?e2 < ?e1 < ?b2) or
(?e1 < ?b2 < ?b1 < ?e2) or (?e1 < ?e2 < ?b1 < ?b2) or
(?b1 = ?b2 and ?e1 = ?e2) or (?b1 = ?e2 and ?e1 = ?b2) ^
?x != ?y
⇒ eoo:overlapsPartiallyWithRange(?x,?y)

# Overlaps totally with range
`RANGE_IDENTIFICATION` ^
`earmark:refersTo(?x,?d) ^ earmark:refersTo(?y,?d) ^
earmark:begins(?x,?b1) ^ earmark:begins(?y,?b2) ^
earmark:ends(?x,?e1) ^ earmark:ends(?y,?e2)` ^

(?b1 <= ?b2 < ?e2 < ?e1) or (?e1 <= ?b2 < ?e2 < ?b1) ^
(?b1 < ?b2 < ?e2 <= ?e1) or (?e1 < ?b2 < ?e2 <= ?b1) ^
(?b1 <= ?e2 < ?b2 < ?e1) or (?e1 <= ?e2 < ?b2 < ?b1) ^
(?b1 < ?e2 < ?b2 <= ?e1) or (?e1 < ?e2 < ?b2 <= ?b1) ^
?x != ?y
⇒ eoo:overlapsTotallyWithRange(?x,?y)

Here, “RANGE_IDENTIFICATION” is a placeholder for the different antecedents to use in case we want to deal with pointer ranges or with XPath pointer ranges. In particular, for the pointer range we have:

`earmark:PointerRange(?x) ^ earmark:PointerRange(?y)`

, and for XPath pointer ranges we have:

`earmark:XPathPointerRange(?x) ^ earmark:XPathPointerRange(?y) ^
earmark:hasXPathContext(?x,?c) ^ earmark:hasXPathContext(?y,?c)`

**Dominance vs. Containment in EARMARK**

In this section we introduce how dominance and containment relations are implemented in EOO, since their intrinsic relation with any kind of overlapping scenario we discuss in the following subsections.

The dominance relation is actually defined by two different and related concepts that have always markup items as subject of dominance assertions. In particular, we say that a markup item A
dominates directly (i.e., \texttt{eoo:dominatesDirectly}) an EARMARK node \( B \) if \( A \) has \( B \) as child. This relation is formally defined as follows:

\[
\text{# Declaration as an object property} \\
\text{eoo:dominatesDirectly} \sqsubseteq \top \\
\text{# Domain} \\
3\text{eoo:dominatesDirectly}.\top \sqsubseteq \text{earmark:MarkupItem} \\
\text{# Range} \\
\top \sqsubseteq 3\text{eoo:dominatesDirectly}.(\text{earmark:Range} \cup \text{earmark:MarkupItem})
\]

The relation between \texttt{eoo:dominatesDirectly} and the parent-child relation in EARMARK\(^{[17]}\) is defined by means of the following SWRL rule:

\[
\text{earmark:MarkupItem}(?x) \land \text{co:element}(?x,?y) \\
\Rightarrow \text{eoo:dominatesDirectly}(?x,?y)
\]

Generalising \texttt{eoo:dominatesDirectly}, we say that a markup item \( A \) dominates (i.e., \texttt{eoo:dominates}) an EARMARK node \( B \) if \( B \) is a descendant of \( A \). This property is transitive and is also a super-property of \texttt{eoo:dominatesDirectly} (i.e., \texttt{eoo:dominatesDirectly} entails \texttt{eoo:dominates}), as defined as follows:

\[
\text{# Declaration as an object property} \\
\text{eoo:dominates} \sqsubseteq \top \\
\text{# Sub-property declaration} \\
\text{eoo:dominatesDirectly} \sqsubseteq \text{eoo:dominates} \\
\text{# Transitivity} \\
\text{eoo:dominates} \circ \text{eoo:dominates} \sqsubseteq \text{eoo:dominates}
\]

The containment is a transitive relation (i.e., \texttt{eoo:contains}) that is defined on the basis of the dominance relation and applies among any EARMARK node (either markup item or range). In particular, we say that an EARMARK node \( A \) contains another EARMARK node \( B \) when one of the following conditions holds:

1. \( A \) dominates \( B \);
2. if \( A \) and \( B \) are markup items, the leaf nodes dominated by \( A \) are a super-set of the leaf nodes dominated by \( B \);
3. if \( A \) and \( B \) are ranges, \( A \) overlaps totally with \( B \) (cf. section “Overlapping of ranges”) and the interval defined by \( A \) contains completely the locations of \( B \).

This relation is thus formally defined as follows:

\[
\text{# Declaration as an object property} \\
\text{eoo:contains} \sqsubseteq \top \\
\text{# Domain} \\
3\text{eoo:contains}.\top \sqsubseteq \text{earmark:Range} \cup \text{earmark:MarkupItem} \\
\text{# Range} \\
\top \sqsubseteq 3\text{eoo:contains}.(\text{earmark:Range} \cup \text{earmark:MarkupItem}) \\
\text{# Transitivity} \\
\text{eoo:contains} \circ \text{eoo:contains} \sqsubseteq \text{eoo:contains}
\]

In addition to that, by means of rule 1, we can also state that the dominance relation is actually a sub-relation of the containment relation (meaning that if \( A \texttt{eoo:dominates} B \), then \( A \texttt{eoo:contains} B \) holds as well), as shown as follows:
While we cannot specify in any way (neither in OWL nor SWRL) the constraint introduced in rule 2, we can define a particular SWRL rule to handle the constraint introduced in rule 3:

```swrl
\( \exists \text{D: coa} \equiv \text{coa:cy} \)
```

### Overlapping of markup items

The case of overlapping markup items (i.e., linked through the symmetric property `oom:overlapsWithMarkupItem`) is slightly more complicated than range overlaps. We define that two markup items A and B overlap when at least one of the following scenarios holds:

1. a markup item A contains a range that overlaps with another range contained by a markup item B;
2. two markup items A and B contain at least a range in common;
3. two markup items A and B contain at least a markup item in common.

The property `oom:overlapsWithMarkupItem` is defined as follows:

```swrl
# Sub-property declaration
oom:overlapsWithMarkupItem ⊆ oom:overlapsWith

# Domain
\( \exists \text{oom:overlapsWithMarkupItem.\top ⊆ earmark:MarkupItem} \)

# Range
\( \top ⊆ \{\text{oom:overlapsWithMarkupItem.earmark:MarkupItem} \}

# Symmetry
oom:overlapsWithMarkupItem ≡ oom:overlapsWithMarkupItem-
```

The three aforementioned scenarios correspond to three different symmetric sub-properties of `oom:overlapsWithMarkupItem`.

The first scenario – i.e., A contains a range that overlaps with another range contained by B – refers to markup items overlapping by range. In the example in Figure 2, the element `ll` overlaps by range with the element `unit`. This is captured by a subproperty of `oom:overlapsWithMarkupItem`, property `oom:overlapsByRange`, that is formally described as follows:

```swrl
# Sub-property declaration
oom:overlapsByRange ⊆ oom:overlapsWithMarkupItem

# Domain
\( \exists \text{oom:overlapsByRange.\top ⊆ earmark:MarkupItem} \land \)
\( \exists \text{oom:dominatesDirectly,(oom:overlapsWithRange.earmark:Range) \}

# Range
\( \top ⊆ \{\text{oom:overlapsByRange.(earmark:MarkupItem} \land \)
\( \exists \text{oom:dominatesDirectly,(oom:overlapsWithRange.earmark:Range) \}

# Symmetry
oom:overlapsByRange ≡ oom:overlapsByRange-
```
The second scenario – i.e., $A$ and $B$ contain at least one shared range – refers to markup items overlapping by content hierarchy. In the example in Figure 2, the element $I2$ overlaps by content hierarchy with the element $unit2$. The corresponding subproperty $eoo:overlapsByContentHierarchy$ is formally described as follows:

```
# Sub-property declaration
eoo:overlapsByContentHierarchy ⊆ eoo:overlapsWithWithMarkupItem

# Domain
∃eoo:overlapsByContentHierarchy.⊑ ⊆ earmark:MarkupItem ∩ ∃eoo:dominatesDirectly.earmark:Range

# Range
⊑ ⊆ ∃eoo:overlapsByContentHierarchy.(earmark:MarkupItem ∩ ∃eoo:dominatesDirectly.earmark:Range)

# Symmetry
eoo:overlapsByContentHierarchy ≡ eoo:overlapsByContentHierarchy-
```

The third scenario – i.e., $A$ and $B$ contain at least another markup item in common – refers to markup items overlapping by markup hierarchy. In the example in Figure 2, the element $lg$ overlaps by markup hierarchy with the element $q$. The related subproperty $eoo:overlapsByMarkupHierarchy$ is formally described as follows:

```
# Sub-property declaration
eoo:overlapsByMarkupHierarchy ⊆ eoo:overlapsWithWithMarkupItem

# Domain
∃eoo:overlapsByMarkupHierarchy.⊑ ⊆ earmark:MarkupItem ∩ ∃eoo:dominatesDirectly.earmark:MarkupItem

# Range
⊑ ⊆ ∃eoo:overlapsByMarkupHierarchy.(earmark:MarkupItem ∩ ∃eoo:dominatesDirectly.earmark:MarkupItem)

# Symmetry
eoo:overlapsByMarkupHierarchy ≡ eoo:overlapsByMarkupHierarchy-
```

The following SWRL rules allows us to catch the constraints of this kind of overlap by inferring the right overlapping relation according to the aforementioned three scenarios:

```
# overlaps by range
earmark:MarkupItem(?a) ⊓ earmark:MarkupItem(?b) ⊓
earmark:Range(?r1) ⊓ earmark:Range(?r2) ⊓
eoo:dominatesDirectly(?a,?r1) ⊓ eoo:dominatesDirectly(?b,?r2) ⊓
eoo:overlapsWithRange(?r1,?r2) ⊓
?a != ?b ^ ?r1 != ?r2
⇒ eoo:overlapsByRange(?a,?b)

# overlaps by content hierarchy
earmark:MarkupItem(?a) ⊓ earmark:MarkupItem(?b) ⊓ earmark:Range(?r) ⊓
eoo:dominatesDirectly(?a,?r) ⊓ eoo:dominatesDirectly(?b,?r) ⊓
?a != ?b
⇒ eoo:overlapsByContentHierarchy(?a,?b)

# overlaps by markup hierarchy
earmark:MarkupItem(?a) ⊓ earmark:MarkupItem(?b) ⊓ earmark:MarkupItem(?x) ⊓
eoo:dominatesDirectly(?a,?x) ⊓ eoo:dominatesDirectly(?b,?x) ⊓
?a != ?b != ?x
⇒ eoo:overlapsByMarkupHierarchy(?a,?b)
```

Approaching inferences through reasoners
The EARMARK Overlapping Ontology can be used by OWL reasoners such as Pellet\cite{18} \cite{33} in order to identify all the possible kinds of overlapping scenarios that happen within any EARMARK document. As an example, running such reasoner according to EOO on the EARMARK file describing the document in Figure 2\cite{19}, we obtain a full and complete description of all kinds of overlaps existing in such document\cite{20}.

In particular, the reasoner identified:

- all the dominance relations among elements that exist in the document, as well as all the related containment relations entailed by dominance;
- that the range “Of Mans First Disobedience, and the Fruit” (r1 from now on) overlaps with the range “the Fruit Of that Forbidden Tree” (r2 from now on), and r2 overlaps with the range “Of that Forbidden Tree” (r3 from now on). Specifically, r1 overlaps partially with r2, and r2 overlaps totally with r3;
- about the last total range overlap, that r2 actually contains r3 and, consequently, the markup items syntax and unit1 contain r3;
- that the markup items in the pairs l1 - unit1, l2 - unit1, l2 - unit2, l3 - unit2, and l4 - q overlap between them. Specifically, the markup items in the first two pairs overlap by range, while those in the following two pairs overlap by content hierarchy, and the last two overlap by markup hierarchy.

Of course, this inference process can be run on any EARMARK document. However, the bigger the document (in terms of the number of OWL assertions that specify the markup structure), the longer it takes for the reasoner to infer those data. For this reason, in some cases, it could be preferable to express as SPARQL 1.1 inserts \cite{17} some of the inference rules that we have shown here as OWL logical axioms and SWRL rules. For instance, the rule specified for identifying the overlaps by markup hierarchy could improve the efficiency of the system if expressed in SPARQL as follows:

```
# Rule 'overlaps by markup hierarchy' in SPARQL
CONSTRUCT { ?a eoo:overlapsByMarkupHierarchy ?b }
WHERE {
    ?a a earmark:MarkupItem ;
    eoo:dominatesDirectly ?x .
    ?b a earmark:MarkupItem ;
    eoo:dominatesDirectly ?x .
    ?x a earmark:MarkupItem .
}
```

According to our experience, this approach considerably reduces the time to infer the existing overlapping scenarios in an EARMARK document, even if it needs to implement manually all the inferences that are needed, including those derived from any ontological axiom, e.g., subsumption, property characteristic (transitivity, irreflexivity, symmetry, etc.), and so on.

**Conclusions**

For EARMARK to be able to claim to be a one-stop answer to overlapping needs of markup authors, we still needed a way to identify when, indeed, ranges and markup items actually overlap.

EARMARK *per se*, in fact, does not have a way to identify overlapping situations, simply allowing them to exist and each overlapping item to ignore the others. With the EARMARK Overlapping Ontology, on the other hand, it is now possible to identify and qualify explicitly every overlapping situation we encounter. For instance in \cite{1} we provide a brief overview of situations and contexts where EARMARK can and has been used, especially in the domain of Digital Humanities.
Also, technically, EARMARK is a stand-off notation, and as such it suffers from the same limitations that all stand-off notations suffer: namely, whenever the source document (the docuverse) is modified outside of the control of the author of the EARMARK annotations, they may (and often will) have the pointers become outdated and wrong. Also in [1] we provide some mechanisms through which EARMARK pointers can be resynchronized with a modified source, that should be able to handle some of the possible situations.

EARMARK still has not finished evolving. The FRETTA parser [2], that provides a way for converting EARMARK documents into XML, and expressing overlapping situations choosing parametrically one of the many existing XML tricks such as fragmentation, milestones or twin documents, is working and complete, but the opposite converter, the one that generates an EARMARK document from an XML file that uses XML tricks to express overlaps, is still to be completed. Once this is finished, we will have a complete solution to the problem of expressing any markup document with Semantic Web technologies, and we will be able to cover all possible situations of conversion of overlapping documents.

Bibliography


[1] This non-implicitness of properties in EARMARK results, of course, in having a linearisation of standard documents (i.e., those that do not contain any overlap) that is more verbose than that obtained by using other markup languages such as XML or TexMecs, both in terms of bytes [31] as well as in terms of comparing RDF statements in an EARMARK document vs. number of markup nodes in an XML document [12]. However, while the gap in bytes between XML-based formats (e.g., ODT used by OpenOffice and OXOMXML used by Microsoft Word) and EARMARK (linearised in Turtle) seems to be proportional when additional overlapping elements are introduced in documents [31], the gap between number of statements and number of nodes changes in favour of EARMARK [12] the more overlapping markup item are added to the document.

As introduced in [12], from a pure syntactical point of view, EARMARK is nothing but yet another standoff notation, where the markup specifications point to, rather than contain, the relevant substructure and text fragments. Thus it is affect of all the usual problems of any standoff notations:

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.

However, EARMARK provides also a number of workarounds to most of the above-mentioned
issues, as discussed in [12].

[2] In our past works on EARMARK, we show how a correct use of Semantic Web technologies can
allow us to query and validate EARMARK documents in a proper way, even simplifying some of
these tasks when overlapping scenarios exist in a document. In particular, in [12] and [31] we show
how a simple natural language query like "give me the textual content of all paragraphs inserted by
John Smith" is very complex to handle by using XPath on XML documents (stored according to both
ODT and OOXML formats) while it is quite trivial by applying SPARQL on EARMARK
documents. Similarly, the syntactic validation of documents with overlapping markup is not so
straightforward to check in XML documents, since it is not easy to retrieve each hierarchy that a
document defines by means of overlapping workarounds (e.g., milestones and fragmentation
elements). However, in EARMARK this task is simplified since all the hierarchies are explicitly
defined without using any workaround and the document validity can be verified easily through a
reasoner against a grammar implemented as an OWL ontology, as we show in [13]. In addition, the
use of OWL allows us to perform also the semantic validation of the markup in EARMARK
documents, with several application in real-case scenarios such as semantic search in digital libraries
and the quality evaluation of legal drafting [30].

[3] as per meaning #3 of the entry on definitive in the Merriam-Webster Dictionary at

[4] In the first paper that deals with overlap in digital texts, in 1988 Barnard et al argue that “SGML
can successfully cope with the problem of maintaining multiple structural views”, and that the
solutions “can be made practical” by means of simple mechanisms, such as by exploiting the
CONCUR feature of SGML [3].

[5] In a famous paper [9], Renear et al. defend their OCHO thesis stating that “If you treat texts as
ordered hierarchies of content objects many practical advantages follows, but not otherwise. Therefore
texts are ordered hierarchies of content objects”.

[6] It's worth noting that many slightly different types of milestones have been proposed: for example,
another (more general) type of milestone consists in using milestone elements to mark the boundary
between sections of a text, as indicated by changes in a standard reference system (e.g. the structure
of pages in a standard codex). In those cases, each milestone element (except the first and the last)
represents both the end of the previous feature and the beginning of the next one.

entities defined in it, while the prefix co refers to entities – used in the EARMARK Ontology –
defined in the old version of the Collections Ontology [6].

[8] Graffoo is a graphical notation for OWL ontologies and it is available at
http://www.essepuntato.it/graffoo.

[9] In the following descriptions the prefix co is used to indicate entities taken from version 1.2 of the
Collections Ontology [6], an imported ontology used for handling collections, available at
http://swan.mindinformatics.org/ontologies/1.2/collections.owl.

[10] General identifier actually refers to the SGML generic identifier, i.e., the SGML term for the local
name of the markup item, e.g., “p” for markup element “<p>...</p>”.
This and all the following excerpts are defined in Turtle [32].

EARMARK Overlapping Ontology: http://www.essepuntoato.it/2011/05/overlapping. The prefix eeo refers to entities defined in it.

OWL 2 DL [MotikOWL2] is based on a particular description logic (DL), i.e., SROIQ [20]. In this paper, we decided to use DL notation for the sake of clarity, instead of adopting one of the possible linearisation of OWL made available by the W3C. We recommend the reading of [25] for more information about DL notation. As an extension of common DL notation, we are using T and Top to indicate the top class and the top object property respectively.

Any OWL 2 DL ontology can be accompanied by SWRL rules so as to guarantee additional inferences that are not directly handled by current ontological definitions. All these rules will be defined using an informal human readable syntax as introduced in [21], where each rule is represented in the form of “antecedent ⇒ consequent” statements, meaning that if the antecedent is true, then the consequent can be inferred. Both antecedent and consequent are a list of ontological assertions separated by “^”. Each assertion can be composed by an atomic entity (e.g., a class or a property) containing zero, one or two variables (each beginning with a “?”) depending on the kinds of unary (i.e., class) or binary (i.e., property) entity used, or by a (boolean, cardinality, etc.) restriction of multiple entities.

Note that OWL 2 DL does not support the unique name assumption typical of database systems. Among the various consequences of this choice, in this case it means that two different IRIs cannot be guaranteed to refer to two different resources.

In the following examples, we introduce some generic SWRL rules for ranges that actually work fully only with instances of the class earmark:PointerRange, which is one kind of range defined in EARMARK. In particular, note that if we consider individuals of the class earmark:XPathRange, the XPath context (defined through the property earmark:hasXPathContext) must be taken into account to identify when such ranges overlap between them. Even if the SWRL rules for XPath ranges are not introduced in this paper for the sake of clarity, in EOO the issue of using also the property earmark:hasXPathContext in such rules has been approached in the most lazy way, saying that two XPath ranges have the same context when the XPath expressions specified are exactly the same. However, currently EOO does not handle the cases of having different XPath expressions that are either semantically-equivalent (i.e., “//p” and “//element()[name() = ‘p’]”) or functionally-equivalent (i.e., they return the same sequence of items).

As anticipated in section “EARMARK”, note that in EARMARK any parent-child relationship between a markup item and a node is defined through the property co:element in case the markup item is defined as a set (i.e., co:Set) or a bag (i.e., co:Bag), while it is defined by the chain co:item o co:itemContent if the markup item is defined as a list (i.e., a co:List). However, the new version of the Collections Ontology [6], available at http://purl.org/co, defines the property co:element as sub-property of the aforementioned property chain, meaning that if we have “A co:item I” and “I co:itemContent B”, then “A co:elements B” holds as well. Even if EARMARK is still using the old version of the Collection ontology, that does not includes the above sub-property axiom, we have added such axiom in EOO in order to map co:element assertions between markup items and nodes as parent-child relationships.

Pellet, OWL 2 reasoner for Java: http://clarkparsia.com/pellet/


An OWL file containing all the assertions about overlaps inferred by the reasoner is available online at http://www.essepuntoato.it/2014/balisage/earmark-overlapping.ttl.

Author’s keywords for this paper: EARMARK Overlapping Ontology; EARMARK; overlapping with range and markup item; dominance vs. containment
Silvio Peroni
<silvio.peroni@unibo.it>
Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

Silvio Peroni holds a Ph.D. degree in Computer Science and he is a post-doc at the University of Bologna. He is an expert in document markup and semantic descriptions of bibliographic entities using OWL ontologies. He is one of the main developers of SPAR (Semantic Publishing and Referencing) Ontologies (http://purl.org/spar) that permit RDF descriptions of bibliographic entities, citations, reference collections and library catalogues, the structural and rhetorical components of documents, and roles, statuses and workflows in publishing. Among his research interests are Semantic Web technologies, markup languages for complex documents, design patterns for digital documents and ontology modelling, and automatic processes of analysis and segmentation. In particular, his recent works concern the empirical analysis of the nature of citations, the study of visualisation and browsing interfaces for semantic data, and the development of ontologies to manage, integrate and query bibliographic information according to temporal and contextual constraints.

_Balisage Series on Markup Technologies_