Linear Extended Annotation Graphs

Vincent Barrellon
Univ Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205
Villeurbanne, France F-69621
firstname.lastname@insa-lyon.fr

Pierre-Edouard Portier
Univ Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205
Villeurbanne, France F-69621
firstname.lastname@insa-lyon.fr

Sylvie Calabretto
Univ Lyon, INSA-Lyon, CNRS, LIRIS, UMR5205
Villeurbanne, France F-69621
firstname.lastname@insa-lyon.fr

Olivier Ferret
Univ Lyon, Lyon 2, CNRS, IHRIM, UMR5317
Lyon, France F-69365
firstname.lastname@univ-lyon2.fr

ABSTRACT

Multistructured (M-S) data models were introduced to allow the expression of multilevel, concurrent annotation. However, most models lack either a consistent or an efficient validation mechanism. In a former paper, we introduced extended Annotation Graphs (eAG), a cyclic-graph data model equipped with a novel schema mechanism that, by allowing validation "by construction", bypasses the typical algorithmic cost of traditional methods for the validation of graph-structured data. We introduce here LeAG, a markup syntax for eAG annotations over text data. LeAG takes the shape of a classic, inline markup model. A LeAG annotation can then be written, in a human-readable form, in any notepad application, and saved as a text file; the syntax is simple and familiar – yet LeAGs propose a natural syntax for multilayer annotation with (self-) overlap and links. From a theoretical point of view, LeAG inaugurates a hybrid markup paradigm. Syntactically speaking, it is a *full* inline model, since the tags are all inserted along the annotated resources; still, we evidence that representing independent elements' co-occurring in an inline manner requires to make the annotation rest upon a notion of chronology, that is typical of stand-off markup. To our knowledge, LeAG is the first inline markup syntax to properly conceptualize the notion of elements' accidental co-occurring, that is yet fundamental in multilevel annotation.

CCS CONCEPTS

•Applied computing → Annotation; •Information systems → *Data model extensions;* **•Theory of computation** → *Data structures design and analysis;*

KEYWORDS

Multistructured data; Markup models.

ACM Reference format:
Vincent Barrellon, Pierre-Edouard Portier, Sylvie Calabretto, and Olivier Ferret. 2017. Linear Extended Annotation Graphs. In *Proceedings of ACM Document Engineering, Malta, September 2017 (DocEng2017),* 10 pages.
DOI: 10.475/123_4

DocEng2017, Malta

DOI: 10.475/123_4

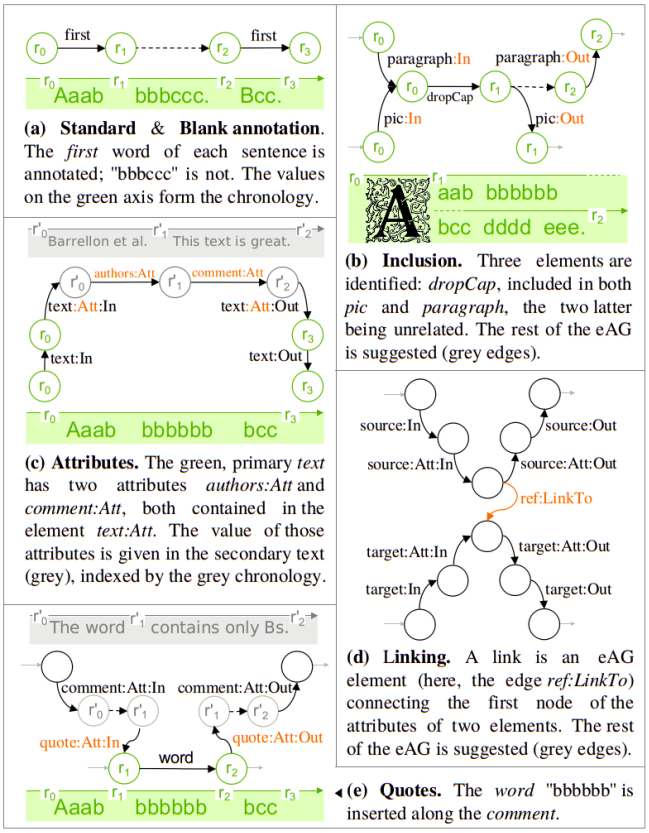
1 INTRODUCTION

The emergence of Digital Humanities has lead to the development of a great number of digital scholarly publishing projects. Most favour the well-known XML-TEI annotation language for transcription and critical enrichment. Indeed, the TEI provides the scholar with an extremely well-documented schema [10], broad enough to fit almost any kind of primary document, and benefits from the assets of XML languages: it is extensible, can be queried, validated and transformed easily. The XML-TEI thus appears as the go-to technology for the editing scholar today.

Yet, editorial criticisms [26] apart, the TEI-XML language suffers from strong formal limitations, inherent to the XML model. In practice, trees are known not to fit some quite common textual description patterns [10, 20, 23]. In particular, XML does not handle overlapping elements, which is an obstacle towards multi-level [34] annotation; additionally, inclusion being represented by nesting in XML (i.e. the location of an element within the scope of another one), there is no way to represent accidental nesting or co-location, that is, the fact two elements occurring at the same place might be independent (and not included one into the other). Inter-elements relations (other than *structural* relations) cannot be represented but by attribute equalities (exemplified by the ID/IDREF mechanism), notoriously hard to restrict by means of a schema [3, 31] and possibly hindering querying [14]. Propositions have been made to conform TEI-XML with more expressive data models [6, 8, 10]; while interesting, those propositions are not compliant with the classic XML tools (XSD, XSLT, etc.) [17].

Some alternative 'multistructured' data models have been proposed to overcome the expressive limitations of XML, by relying on more general directed acyclic graph formalisms than just trees [31], or even cyclic graphs [16] – while maintaining the possibility to validate the data. Yet, acyclic models, if they do allow multilayer annotation, exhibit the same weakness as XML regarding the representation (and hence, the validation) of non-structural relations between elements; cyclic data models, that rely upon RDF, do not benefit from an efficient validation mechanism yet [29, 30].

In a former paper [2], we introduced extended Annotation Graphs (eAG), a cyclic-graph data model experimenting the simulation relation [19] as a validation mechanism. An interesting aspect of simulation is, as we evidenced, that it can be guaranteed *by construction*, enabling to validate cyclic, multistructured data *on the fly*, just like when using grammar-based validators for XML.



(a) Standard & Blank annotation. The *first* word of each sentence is annotated; "bbbccc" is not. The values on the green axis form the chronology.

(b) Inclusion. Three elements are identified: *dropCap*, included in both *pic* and *paragraph*, the two latter being unrelated. The rest of the eAG is suggested (grey edges).

(c) Attributes. The green, primary *text* has two attributes *authors:Att* and *comment:Att*, both contained in the element *text:Att*. The value of those attributes is given in the secondary text (grey), indexed by the grey chronology.

(d) Linking. A link is an eAG element (here, the edge *ref:LinkTo*) connecting the first node of the attributes of two elements. The rest of the eAG is suggested (grey edges).

(e) Quotes. The *word* "bbbbbb" is inserted along the *comment*.

Figure 1: The eAG syntax in a nutshell.

We introduce here LeAG, a markup syntax for textual eAG annotations. LeAG takes the shape of a classic, inline markup model. A LeAG annotation can then be written, in a familiar, human-readable form, in any notepad application, and saved as a text file – yet LeAG offers a natural syntax for overlapping, multilayer annotation.
First, we provide the reader with a quick, example-based summary of the eAG data model. We then introduce the LeAG syntax based on the same examples, so as to make the translation between an eAG and a LeAG clear. We eventually elaborate how a LeAG document can be deterministically parsed into a corresponding eAG.

2 EXTENDED ANNOTATION GRAPHS

Extended Annotation Graphs (eAG) [2] is a schema-aware, stand-off markup model. Following [4], it is based upon the notion of **chronology**. A chronology is an ordered set of reference values that index the data to be annotated (e.g. inter-character positions for texts). An eAG is a rooted, single-leafed, directed and labelled graph whose nodes bear a reference value. Basically, a labelled edge connecting two nodes v_1 and v_2 is a tag put onto the portion of the data delimited by the reference values of v_1 and v_2. The edge and nodes together constitute an **element** – see figure 1.a. The following notions refine this general principle.
Sequential annotation [fig. 1.a] eAG enables to define not only elements, but also *sequences* of elements. An element B *directly* follows an element A iff the end of A and the start of B are one same node. eAG also enables to express *sparse* sequences : the gap between two consecutive elements is filled by a 'blank annotation'[1].
Inclusion [fig. 1.b] It is possible to assess that a sequence S of elements is included in another element A (i.e. that A is constituted of the elements S), by encasing S between two edges *A:In* and *A:Out*. An element can be included in more than one element. This property enables to express multitrees and goddags [28]. It is worth noting that a hierarchy of elements takes the shape of a path: we call them **hierarchical annotation path** (HAP). The spine of an eAG is an acyclic set of HAP sharing elements.
Attributes [fig. 1.c] The attributes of an eAG element X are elements whose labels bear the suffix *:Att*, and that are included in a special element *X:Att* itself included in X. The values of the attributes are not part of the initial, primary corpus: they belong to separate, secondary resources, indexed by a specific chronology.
Links [fig. 1.d] Links are explicit relations between pairs of elements. In eAG, those relations are represented by special elements whose root belongs to the source element, and whose leaf belongs to the target element. The name of the linking element bears the suffix *:LinkTo*. A link can be structured (i.e. contain other elements) or be a single edge. eAG links may induce cycles; yet, they are handled by an SeAG schema just like any other element.
Quotes [fig. 1.e] eAG introduced quoting elements to meet the need of scholars for inserting, inside a commentary, part of an exogeneous, possibly annotated, resource [12]. A quoting element is an attribute whose content is a sequence of elements identified in the primary resources. Quotes necessarily result in cyclic graphs [2] (see p. 7); yet, they too are handled by SeAG schemas.
Apart from the expressive power of this syntax (illustrated below), the eAG model benefits from a **schema language** (SeAG) that manages multistructured, overlapping, cyclic annotation. SeAG validation relies upon the notion of rooted **simulation**[2] [2]. Intuitively, the existence of a simulation of an eAG I_S by a schema S implies that all the paths of I_S starting at its root have a corresponding path in S, whose label sequence is identical. Yet, as shown above, the syntactical structures of an eAG are made out of sequences of elements, i.e., of labelled paths. Thus, S is descriptive of I_S, because any sequence of elements in I_S must have a matching sequence in S. Conversely, S works like a schema: it simulates (validates) the graphs that contain solely sequences of elements it defines.
Figure 2 illustrates SeAG validation. The paths the schema S contains define the set of valid element sequences for the instances (e.g. [*Unit1 - Unit2 - Unit3*]). SeAG also makes use of epsilon edges, or blank annotations, to denote **optional** (e.g. *Unit3* can be bypassed by the ϵ_4 edge, resulting in [*Unit1 - Unit3*]) or **repeatable** elements (e.g. since ϵ_2 defines a cycle, any repetition of *6Letters* is valid).
Importantly, SeAG supports two kinds of multilayering annotation. First, two parallel paths of the schema can be instantiated, independently (i.e. without worrying about overlap), on the same resource (cf. I_a on fig. 2). We call this **schema-based multilayering**. Second, *one* path of the schema can be instantiated *several times* on the same portion of the resources (cf. I_b, same fig.). This **simulation-based multilayering** allows the expression of self-overlapping elements, quite useful in linguistics, as illustrated below.

[1] Represented by dotted, 'epsilon edges' hereafter.
[2] *Node-typed* [2], rooted simulation actually; yet, node types can be omitted here.

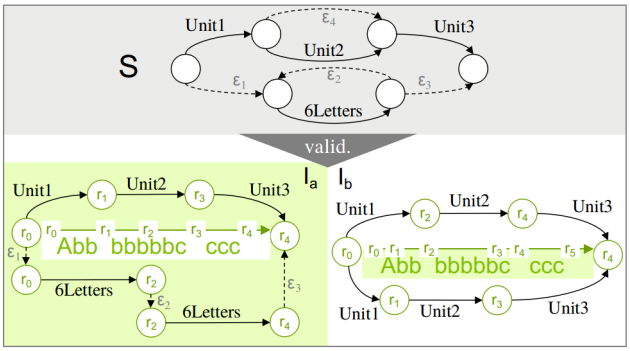


Figure 2: The schema S contains several parallel paths defining the valid element sequences. One element is optional (*Unit2*), one can be repeated (*6Letters*). The eAG I_a instantiates two different paths on the same text. I_b instantiates the same path of the schema *twice*.

3 A RUNNING EXAMPLE

A common linguistic annotation is the identification of anaphoric chains (AC). ACs are sequences of singular expressions so that if one of them refers to something, then they all do [9]. Consider the following text, adapted from *The Village of Ben Suc* by J. Schell:

> An ARVN officer asked a young prisoner questions, and when he failed to answer, beat him. An American observer who saw the beating that happened then reported that the officer "really worked him over". After the beating, the prisoner was forced to remain standing for hours.

One may identify, among others, the following ACs: [a young prisoner / he / him / him / the prisoner], [An American observer who saw the beating], [the beating that happened then / the beating]. Annotating the text in terms of ACs made out of *expressions* is not trivial in XML. Since ACs do not form neither a sequence nor a hierarchy, they cannot be represented as normal, spanning XML elements. The classic solution is to identify only the singular expressions in the text and then relate them together accordingly by their IDs in `<linkgrp>` elements [11]. That solution, apart from being hard to validate, does not represent the fact an AC is *composed of* expressions consistently with the XML syntax. Moreover, it does not extend to this example, which exhibits self-overlap [28][3].
In SeAG/eAG, annotating anaphoric chains is straightforward. Suppose we aim at identifying the ACs and their constitutive expressions, but also to qualify their relative weight, e.g. by reifying the relation 'this AC contains more expressions than that one'. The SeAG schema for that annotation is given on figure 3. That schema defines an *Extract* element containing one, or several parallel, *AC*s, each containing one or more *Exp* elements; also, an *AC* may be the source of a *Longer:LinkTo* link targeting an *AC*. A corresponding eAG (restricted to the ACs relative to *the American observer* and *the beating*) is given on the same figure. Noteworthily, since eAG is a stand-off markup model, overlap is expressed naturally[4].

[3]Cf. *An American observer who saw the beating* and *the beating that happened then*.
[4]Cf. the *Exp* elements ranging from r_1 to r_3 and r_2 to r_4 respectively.

4 LINEAR EXTENDED ANNOTATION GRAPHS

Linear extended Annotation Graphs (LeAG) is an inline markup syntax for eAG. The purpose of LeAG is to enable the expression of eAG annotations by means of any notepad application, in a human-readable form. LeAG must therefore: 1) support unambiguous translation into the eAG syntax, and 2) enable to represent, by means of tags, multilayer, cyclic annotation.
The first part of this paragraph is a theoretical discussion about the hybrid nature of the LeAG markup, between the inline and stand-off paradigms, which will lead to the formulation of an equivalence relation for LeAG documents. We then introduce, step by step, the LeAG syntax: we gradually show how to represent the different bricks eAGs are made of in a markup manner: hierarchies, multitrees (and goddags), attributes, links and quotes. We then interrogate the correspondence between eAG and LeAG, in order to establish the parsability of LeAG into eAG.

4.1 Inline multilayer annotation

Multistructured models are meant to support the simultaneous expression of several annotation paradigms. For instance, one may want to annotate a text by identifying, independently, its *grammatical* (substantive, adjective, etc.) and its *semantic* (proposition, topic, etc.) structures. To achieve that goal, eAG makes a clear distinction between the representation of **inclusion**[5], which is a modelling relation that makes sense within one annotation paradigm, and **nesting** or **co-occurrence**[6], which is a fortuitous situation in which two independent elements occur at the same place. And indeed, the eAG syntax for inclusion is *explicit* (see figure 1.b), while nesting *happens* when two elements X and Y are so that $ref(start(X)) \leq ref(start(Y))$ and $ref(end(Y)) \leq ref(end(X))$ – hence nesting is uniquely defined in terms of reference values.
Yet the notion of chronology is quite impacted by the shift from stand-off to inline markup. In eAG, in order to fit multimedia annotation, several chronologies can be defined, and each node is associated a value from one of those chronologies. In a text-only markup setting, a natural chronology is implied by the text itself: the set of inter-character positions. As a consequence, LeAG rests upon that single, natural chronology, that does not even need to be made explicit: tags are simply inserted, within the text stream to be annotated, at the position a corresponding node of an eAG would have made reference to. E.g., annotating the *substantive* in "Let us garlands bring." is done by inserting a pair of tags as follows: "Let us `[Substantive}`garlands`{Substantive]` bring."
Still, in spite of being considerably simplified compared to eAG, the notion of chronology is still *central* to LeAG, because it is absolutely necessary in order to represent co-occurrence or nesting. Consider the very elementary text stream `ABC`. A chronology for this text is: {*start() = before(A), after(A) = before(B), after(B) = before(C), after(C) = end()*}. Identifying an element Ω between the positions *before*(*A*) and *before*(*C*) is done as follows: `[Ω}AB{Ω]C`. The text stream, since it has been added new characters (the ones that constitute the tags), has been altered by this operation.

[5]E.g. a *proposition* contains a *topic*.
[6]A word may happen to be both a *substantive* and the *topic* of a *proposition*: *topic* and *substantive* co-occur; *substantive* is nested in *proposition*;*topic* is included in *proposition*.

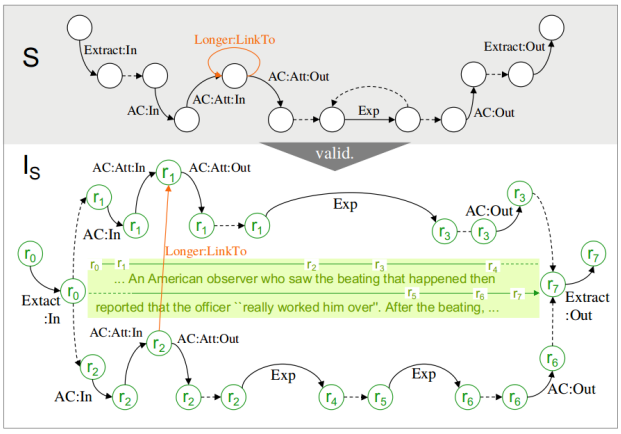


Figure 3: SeAG/eAG illustrating anaphoric chain annotation. An *Extract* contains overlapping AC, composed of overlapping *expressions*. The *Longer:LinkTo* link reifies a relation from the more to the less numerous AC.

Yet, interestingly, even in the annotated text stream, the original chronology is still operative to index a very particular text substream, that is, the text stripped from the tags – i.e. the original stream. This may sound tautological; nonetheless, this remark is fundamental, since this bare text stream is the one an editor will consider when she wants to annotate the corpus independently from any previous annotation – that is, when proceeding to multilayer annotation. Indeed, if the editor wants to identify another element ω, ranging from *before*(A) to *before*(C), she may insert an opening tag at the position *before*(A), and a closing tag at *before*(C), without considering the other tags, resulting in[7] A_1 =[Ω]\[ω]AB{Ω]{ω]C. One may also consider that, in the original text stream, *start()* = *before*(A) – so the annotation A_2 =[ω]\[Ω]AB{Ω]{ω]C (where [ω} is inserted at the position *start()* this time) shall be considered equivalent to A_1. Similarly, since the two elements Ω and ω are independent, the order in which they are identified shall be indifferent: the two opening (closing, respectively) tags in $A1$ and $A2$ can be inverted, resulting in two more equivalent markups: A'_1 = [ω}\[Ω]AB{ω]{Ω]C and A'_2 = [Ω}\[ω}AB{ω]{Ω]C.

Hence the following relation:

Equivalent LeAG. Let us call **trains of tags** the largest sets of tags, in a LeAG, that are not separated by a character from the original text. Two LeAG are **equivalent** iff they differ only by the order of the tags that belong to their respective trains of tags.

This notion of equivalence actually reflects the fact LeAG, though it is an inline markup syntax, rests upon a notion of chronology that is typical of stand-off markup models. Indeed, one way to interpret the above equivalence relation is by saying that in a LeAG, tags only *make reference* to the position they occupy in the original text stream. Surely, two tags making reference to the same position may be written in any order. Since, in practice, two such tags will not be separated by any character from the bare data and constitute a 'train of tags', it follows that in a train of tags, the order in which the tags are written is indifferent.

[7]See paragraph 4.2.2 for the actual syntax for multilayer annotation.

As a consequence, contrary to XML, the nesting of an element B inside the scope of an element A *cannot* be a means to represent the inclusion of B inside A. Thus a syntax is needed to represent inclusion (cf. 4.2.1). Second, since inserting tags does not alter the chronology that indexes the original text, tags can be considered not to take "any room" along that chronology. This suggests that inserting exogenous resources within the primary resources, e.g. structured comments, can be done *inside special tags that open and close at the same position in the original stream* (cf. 4.2.3).

4.2 The LeAG syntax

In the following paragraph, based on the above considerations, we gradually introduce the LeAG syntax. The content of the LeAG tags will be defined by means of formulae in which orange characters are constants and italics denotes variables. (Black) parenthesis are mathematical delimiters, not variables or constants. A **field** is either a variable or a formula enclosed in parenthesis. An optional field is followed by the character ?. A field that can be repeated is followed by +, one that is both optional and can be repeated is followed by *. Concatenation is implicit. Space characters are represented by underscores.

4.2.1 Mono-hierarchy of attribute-less elements. As stated above, some explicit syntax is needed to represent inclusion in a markup model that supports multilayer annotation. This paragraph presents how to express single-layered annotation. The next paragraph extends LeAG towards multilayered annotation.

Elementary spanning elements. An elementary spanning elements (ESE) is the syntactical structure dedicated to the labelling of a section of the primary resources, with the possibility to assess that the current element is included in other elements of the annotation. ESE are represented by a pair of opening and closing tags whose *substance* field has the same value, according to the following:

OTag := [*substance* }
CTag := { *substance*]
substance := *name fathers*? (, _ *ID*)?
fathers := _in_ *context*

Above, *name* is the name of the current element and works as a label on the primary resources enclosed by the pair of tags; *context* provides a designation of the elements that contain the current element[8]. The *ID* field will be discussed in the paragraph 4.2.4.

The **content of an element** is constituted of the tags themselves and the whole text (primary resources + tags) they span over.

Rule 4.2.1 The opening and the closing tags defining one ESE cannot belong to the same train of tags.

Back to the example. In order to identify *one* anaphoric chain in the extract of *The Village of Ben Suc*, it suffices to define three element names *Extract*, *AC* and *Exp* for the identification of the extract, the AC and its constituting expressions respectively, and to build the following pairs of opening/closing tags:

- [Extract} and {Extract] ;
- [AC in Extract} and {AC in Extract], assessing that an AC is included in an extract;
- [Exp in AC} and {Exp in AC], assessing that an expression is included in an AC.

[8]We will see that an element may have more than one father, thanks to the notion of grafts. See paragraph 4.2.2.

The following LeAG L_1 annotates the anaphoric chain regarding the young prisoner accordingly.

L_1 :[Extract}An ARVN officer asked [AC in Extract}[Exp in AC}a young prisoner{Exp in AC] several questions, and when [Exp in AC}he{Exp in AC] failed to answer, beat [Exp in AC}him{Exp in AC]. An American observer who saw the beating that happened then reported that the officer "really worked [Exp in AC}him{Exp in AC] over". After the beating, [Exp in AC}the prisoner{Exp in AC]{AC in Extract]was forced to remain standing for hours.{Extract]

4.2.2 Grafts: Multilayer annotation. We now extend the above syntax to multilayer annotation. Multilayer annotation may occur in two distinct situations: first, the schema defines several annotation paradigms; second, one path of the schema is instantiated several times onto the same resources (figure 2).

The challenge is to make sure that in any case, the tags of a multilayer LeAG shall be unambiguously associated with the layer(s) they are part of. When the set of the elements' names of two co-existing layers do not intersect, assessing to which layer a tag belongs is trivial. At the opposite, simulation-based multilayering, which is prone to self-overlap, will be problematic: in that case, two overlapping elements cannot be discriminated neither on the basis of their name nor by looking at the name of their fathers. Anaphoric chains annotation is a canonical example of such a setting.

For instance, in the excerpt of *The Village of Ben Suc*, consider the ACs relative to *the American observer* and *the beating* respectively. A naïve approach making use of the syntax for single-layered annotation would yield the following annotation – which is faulty:

[Extract}An ARVN officer [...] beat him. [AC in Extract}$_1$[Exp in AC}$_2$An American observer who saw [AC in Extract}$_3$[Exp in AC}$_4$the beating{Exp in AC]$_5${AC in Extract]$_6$ that happened then{Exp in AC]$_7$ reported that the officer "really worked him over". After [Exp in AC}$_8$the beating{Exp in AC]$_9${AC in Extract]$_{10}$, the prisoner was forced to remain standing for hours.{Extract]

Indeed, it is undecidable whether the *Exp* element starting at the tag 4 ends at tag 5 or 7. Moreover, there would be no way to ascertain to which *AC* an *Exp* ranging from tag 4 to tag 5 would belong to. An intuitive disambiguating solution – at least to the human eye – consists in colouring the tags belonging to distinct layers:

[Extract}An ARVN officer [...] beat him. [AC in Extract}$_1$[Exp in AC}$_2$An American observer who saw [AC in Extract}$_3$[Exp in AC}$_4$the beating{Exp in AC]$_5${AC in Extract]$_6$ that happened then{Exp in AC]$_7$ reported that the officer "really worked him over". After [Exp in AC}$_8$the beating{Exp in AC]$_9${AC in Extract]$_{10}$, the prisoner was forced to remain standing for hours.{Extract]

Now it is clear that the element starting at tag 2 ends at tag 5, overlapping with the element starting at tag 4 and ending at tag 7. Importantly, not only have we coloured differently the elements (2-5) and (4-7) in order to make their respective opening and closing tags match, but also have we given a common colour to the elements (3-10), (4-7) and (8-9), which indicates that the two expressions (4-7) and (8-9) belong to the same AC (3-10), for instance.

Grafts. The notion of *grafts* follows the above intuition. Grafts are coloured LeAGs that are anchored onto an existing LeAG. They express, either locally or at the scale of the whole document, some additional enrichment on top of the annotation that has, at a certain point in time, been done already.

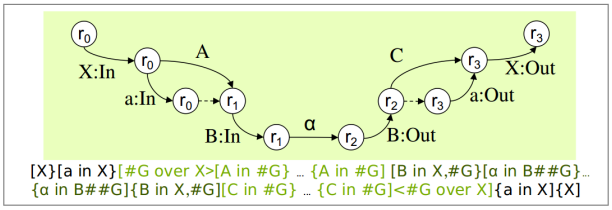


Figure 4: A LeAG and a matching eAG, where an element (B) has two fathers, one in the uncoloured hierarchy, and the other in a graft. The colours of B, namely #G and # (uncoloured), are repeated in the context of its son element α.

Consider the LeAG L_1 at the end of paragraph 4.2.1. L_1 identifies one AC and its constituting expressions (Exp), within an extract. A **graft** must be defined in order to identify, in *the same* extract, another AC, e.g. the AC regarding *the beating*, since this addition will result in a non-hierarchical LeAG. This is done as follows:

(1) The element of the existing annotation that will serve as the *context* of the graft is identified: Extract, here.

(2) A name of 'colour', *nameC*, is defined, in the form:
nameC := # *colour*
where *colour* is a string that identifies the graft, e.g. "#Red".

(3) The range of the graft is specified by inserting, within the frame of the context element, a pair of colour tags:
Otag := [*nameC* _in_ *context* >
Ctag := < *nameC* _in_ *context*]
with the *nameC* and *context* fields as defined above. For instance, the span of the AC regarding "the beating" is the following:
[Extract} An American observer who saw [#Red over Extract>the beating that happened then reported that the officer "really worked [Exp in AC}him{Exp in AC] over". After the beating<#Red over Extract], [...] {Extract]

(4) Then *nameC* serves as a context for the top elements of the graft. Here, one AC element spans over the whole graft:
[Extract} [...] An American observer who saw [#Red over Extract>[AC in #Red}the beating that happened then reported that the officer "really worked [Exp in AC}him{Exp in AC] over". After the beating{AC in #Red]<#Red over Extract], [Exp in AC}the prisoner [...] {Extract]

(5) Elements included in the top elements of the graft are defined, their *context* field keeping record of the colour of the upper element. For instance, here, two Exp belong to the red AC:
[Extract}[...] An American observer who saw [#Red over Extract>[AC in #Red}[Exp in AC#Red}the beating that happened then{Exp in AC#Red] reported that the officer "really worked [Exp in AC}him{Exp in AC] over". After [Exp in AC#Red}the beating{Exp in AC#Red]{AC in #Red]<#Red over Extract], [Exp in AC}the prisoner [...] {Extract]
Similarly, had one Exp element had any child, the context field of the tags defining that element would have been Exp#Red.

Based on that principle, the LeAG L_2 on figure 5 identifies the three anaphoric chains regarding *the prisoner*, *the beating* and *the American observer* respectively – which is a case of simulation-based multilayer annotation with self-overlap.

L_1 :[Extract}An ARVN officer asked [Exp in AC}a young prisoner{Exp in AC] questions, and when [Exp in AC}he{Exp in AC] failed to answer, beat [Exp in AC}him{Exp in AC]. [#Blue over Extract>[AC in #Blue][Exp in AC#Blue}An American observer who saw [#Red over Extract>[AC in #Red][Exp in AC#Red}the beating<#Blue over Extract]{Exp in AC#Blue]{AC in #Blue] that happened then{Exp in AC#Red] reported that the officer "really worked [Exp in AC}him{Exp in AC] over". After [Exp in AC#Red}the beating{Exp in AC#Red]{AC in #Red]<#Red over Extract], [Exp in AC}the prisoner{Exp in AC]{AC in Extract]was forced to remain standing for hours.{Extract]

Figure 5: Three-layered LeAG.

Complements. (1) Grafts are added on top of an existing annotation spanning over the whole document. Before the first graft is defined, the annotation has to be hierarchical[9]. Thus we can refer to this underlying hierarchical annotation as the **uncoloured hierarchy** of a LeAG. Tags of this hierarchy either have no explicit colour or, when they also belong to a coloured graft, the colour of that graft plus a 'blank' colour, # – see element α in figure 4. (2) A graft may be defined either on the underlying hierarchy (figure 4) or on an element from another graft. (3) An element may have several fathers, belonging to grafts or to the uncoloured hierarchy indifferently (cf. element B, figure 4). (4) The span of the graft shall not necessarily equal the one of its context element (figure 5).

4.2.3 Standard inserts: Attributes; structured comment. So far, we have seen how to label the primary resources by means of entangled hierarchies of elementary spanning elements. Still, editing is not only about labelling: sometimes, additional, structured information must be added on top of the labels. In XML, this kind of information constitutes elements' attributes; still, adding attributes to an element is like annotating the element itself, that is, for the editor, inserting secondary, structured data that does not bear on the primary resources but on the tags.

Similarly, providing the editor with means to express critical information, not by labelling the primary resources, but by inserting assertions is a useful feature. Introductions, comments and punctual notes, in their digital form, fall into that category of annotations. Attributes and punctual comments share the property of not being expressible with elementary spanning elements. In LeAG, both will be represented by means of **inserts**. An insert is similar to void elements in XML in that: (1) it is both opening and closing, which means, in the LeAG vocabulary, that inserts start and end at the same position; (2) it is self-contained, in the sense that the tag representing the insert is the insert's content.

Attribute insert: general syntax. The syntax of an attribute insert respects the following formula:

$$\mathit{InsertA} := \texttt{[Att\_of\_}\ \mathit{context}\ \texttt{\_;\_}\ \mathit{LeAG}\ \texttt{]}$$

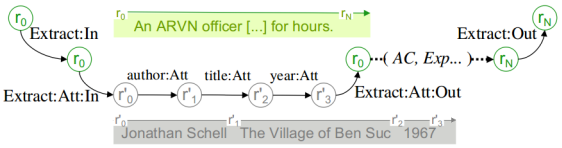
where *context* is the coloured name of the element whose attributes are described in the insert, and *LeAG* is some structured data conform to the LeAG model, constituting the content of the attributes.

[9]This is not a tough constraint, since a single element spanning over the whole corpus is an elementary hierarchical annotation.

Attribute insert: example. So far, the passage of *The Village of Ben Suc* as a whole was simply labelled as an `Extract`. The following LeAG provides, as attributes of the `Extract`, the author's name, the title and the publication year of the novel:

`[Extract}[Att of Extract ; [author}Jonathan Schell {author][title}The Village of Ben Suc{title][year}1967 {year]]`An ARVN officer, [...] for hours.`{Extract]`

The insert corresponds, in eAG, to a hierarchy of elements bearing the suffix `:Att`, included in the element `Extract`:



In the LeAG, there is no need neither to specify the `:Att` suffix for the elements defined inside the attribute insert, nor to indicate in the context field of the top elements among them, that they are included in the insert[10]. The same applies to comment inserts:

Comment inserts: general[11] syntax. The general syntax of a comment insert is the following:

$$\mathit{InsertC} := \texttt{[}\ \mathit{name}\ \texttt{:Att\_in\_}\ \mathit{context}\ (\ \texttt{,\_}\ \mathit{ID})\ ?\ \texttt{\_;\_}\ \mathit{LeAG}\ \texttt{]}$$

where *name* is the name of the insert, *context* is the coloured name of the elements the insert is the son of and *LeAG* structured data conform to the LeAG model, constituting the content of the comment. The *ID* field will be discussed in the paragraph 4.2.4.

Comment insert: an example. The following LeAG incorporates a *comment* regarding the context of *The Village of Ben Suc*:

`[Extract}`An ARVN`[Comment:Att in Extract ; [Att of Comment ; [authorOfComment}Barrellon et al.{authorOfComment]]The mention of the [acronym}ARVN{acronym] refers to the Vietnam War.]` officer asked [...] for hours.`{Extract]`

A comment being an element, it may possess attributes, as illustrated above (e.g. to specify the name of its authors).

Inserts in a train of tags. The case of inserts within a train of tags has to be discussed. Consider the LeAG `[A}...{A][B in A ;` *LeAG*`][A}...{A]`. In the absence of a schema, it is not possible to assess to which *A* element *B* belongs. If there is a schema that does not restrict the position of the element *B* either at the beginning or at the end of the element *A*, neither.

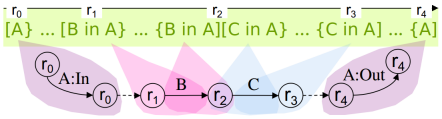
Second, consider `[A}...[B in A ;` L_1`][C in A ;` L_2`]...{A]`. The LeAG itself is not ambiguous: it states that the inserts *B* and *C* occur at the same position. Yet, in the perspective of parsing the LeAG into an eAG (cf. paragraph 6), since in the corresponding eAG, two inserts will form a sequence, there is no indication in the LeAG about which insert will come first. The following conventional rule clarifies those situations:

Rule 4.2.3 When there is no schema or when the schema does not clarify the following situations, it shall be considered that (1) when an insert occurs in a train of tags where an opening and a closing tags identically match the context field of the insert, then the insert conventionally belongs to the *opening* element; (2) when two inserts with the same context field occur in the same train of tags, the alphabetical order between the tags considered as strings provides a conventional order between the inserts.

[10]*Id est*, there is no need to write `[author in Att of Extract}`, for instance.

[11]A refinement of the following syntax will be proposed in the paragraph 4.2.4.

4.2.4 Links and Quoting elements. The last aspect of eAGs that needs to be translated into LeAG is links or quotes. We have seen that in eAG, links and quotes are expressed harmoniously with the other elements (i.e. by means of nodes and edges) and, for that reason, can be properly *validated*. In particular, compared to XML where a link is but an ID/IDREF pair, in SeAG/eAG, the nature of the two elements connected by a link can inherently be restricted. Still, since links and quotes denote distant connections across the corpus that may result in cyclic annotations (i.e. along the text stream, the beginning of an element comes after its end), it is not possible to represent them by means of pairs of tags along the text stream. Thus, LeAG makes use of an additional feature: the ID field. ID fields work as an identifier of either the source or the end of a connection (link/quote), hence enabling to position the extreme *nodes* of such elements inside the LeAG, that is, to position the elements themselves. Yet, ID fields are not *tag* identifiers. Indeed, regardless of the parsing strategy adopted, there is no one-to-one correspondence between the *tags* of a LeAG and the *nodes* of an eAG expressing the same annotation, as evidenced below[12]:

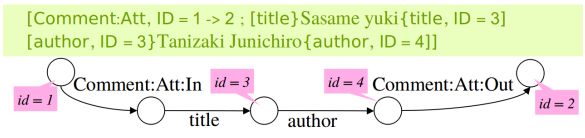


Indeed, because the element *A* contains other elements, the tag `[A}` translates into *two* nodes whose reference values point towards the position of `[A}` inside the document, connected by an edge `A:In`, while the tag `[B}` relates to *one* node only. Conversely, two tags may relate to the same node: since the element *C* starts where *B* ends, both `{B]` and `[C}` relate to the node that separates *B* and *C*. Yet, a finer correspondence between the LeAG tags and a *subset* of the nodes of the corresponding eAG can be exploited for expressing links and quotes: (1) an opening tag positions (and hence, matches) the root of the corresponding element in the eAG; (2) a closing tag positions the leaf of the corresponding element in the eAG; (3) an insert positions both the root and the leaf of the corresponding element in the eAG. ID fields exploit that connection, as follows.

ID fields. Since opening and closing tags of ESE relate to either the root or the leaf of an element in the corresponding eAG, ESE *ID* fields contain a singleton value *K*. *A contrario*, an insert *ID* shall possibly designate the root and the leaf of the corresponding eAG element and thus contains a pair of values *M* and *N*:

ID := `ID_=_` *K* (singleton syntax)
ID := `ID_=_` *M* `_->_` *N* (pair syntax)

Basic example. Let us consider the following comment and a matching eAG (pink flags represent the node identifiers):

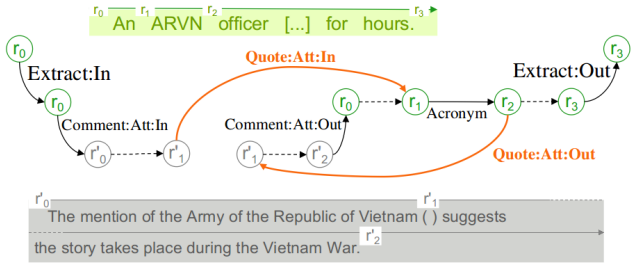


Noteworthily, the relation between ID values and root/leaves nodes is only *surjective*. Thus, the ID of the closing tag of an element and that of an element that immediately follows have to be the equal (e.g. `{title, ID = 3]` and `[author, ID = 3}`, above).

[12]Colours relate the eAG nodes / edges to the tags that define their position and label.

The syntax for links and quotes are based on that mechanism – plus some improvements on the insert syntax.

Quote elements. Quote elements enable to include an element identified in the primary resources within a comment. In eAG, quoting the ARVN acronym from the extract of *The Village of Ben Suc* within a comment can be done as follows:



The two (orange) edges permit to *structurally include* the quoted element inside the comment element.

In LeAG, since the content of a comment has to be written inside the insert itself, quoting, inside the *LeAG* field of a comment, an element that has been identified elsewhere in the annotation cannot be done but by reference. Therefore, quoting elements appear as special comment inserts, whose *LeAG* field has been replaced by an *ID* field (with the pair syntax):

Quote := `[` *name*(`_in_` *context*)?(`,_` ID_1)? `_;_` ID_2 `]`
ID_i, $i \in \{1,2\}$:= `ID_=_` M_i `_->_` N_i

The pair of values of the ID_2 field must then refer to some tag(s) somewhere else in the LeAG that delimit either an element or a sequence of elements.

Quote: example The LeAG representing the above eAG is:

```
[Extract}[Comment:Att in Extract ; The mention of the
Army of the Republic of Vietnam ([Quote ; ID = 1-> 2])
refers to the Vietnam War.]An [Acronym in Extract, ID =
1}ARVN{Acronym in Extract, ID = 2] officer [...] hours.{Extract]
```

This LeAG does correspond to the eAG above, since it states that the *Extract* contains a *Comment:Att*, made out of some not annotated text (which translates into an epsilon edge), followed by a *Quote* containing an annotation graph whose root and leaf have the identifiers '1' and '2' respectively; *Extract* further contains an *Acronym*, whose root and leaf identifiers are '1' and '2' respectively.

Links. An eAG link is an element whose root is a node from an element and whose leaf is a node from another element.

First, to represent such a graph in LeAG, we need to be able to identify a node *inside* any element. Consider the link in figure 3. It connects the internal nodes of two *AC:Att* elements that contain nothing but those nodes. Yet, the *ID* fields of an insert with no *LeAG* field, suit to represent those *AC:Att* elements, only identifies the root and leaf of the matching element, not an internal node. To fill this gap, we define **void inserts**:

VoidInsert := `[ in_` *context* `,_` *ID* `]`
ID := `ID_=_` *N*

Such an insert neither has a name nor a *LeAG* field, but it does have a context (the element it is included in) and an *ID* field. Placed immediately after an opening tag, e.g. `[A}`, a void insert `[in A, ID = 1]` enables to give the identifier '1' to a node that, in the corresponding eAG, is the node ending the *A:In* edge.

Second, we need a means to express that an element may start inside an element and end inside another one. For such a special element, we defined **link insert**:

Link := `[` *name* `:LinkTo_in_` *context* `,_` *ID*(`_;_` *LeAG*)? `]`

ID := `ID_=_` *M* `_->_` *N* OR `ID_=_->_` *N*

The *LeAG* field defines the content of the link; if empty, the link is an edge. The leaf of the link, identified by the value of the variable *N* above, must be an internal node of some element, represented elsewhere by a void insert.

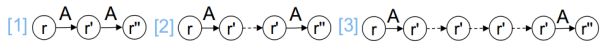
Link: example. Figure 3 illustrates how to annotate different, overlapping AC in an extract, and how links could reify an order relation between them. The following LeAG expresses the same annotation, extended to three ACs (*the prisoner*, *the beating*, *the American observer*) as in the eAG on figure 3:

```
[Extract}An ARVN officer asked [AC in Extract}[AC:Att of AC ; [Longer:LinkTo, ID = -> 2][Longer:LinkTo, ID = -> 1]][Exp in AC}a young prisoner{Exp in AC] questions, and when [Exp in AC}he{Exp in AC] failed to answer, beat [Exp in AC}him{Exp in AC]. [#Blue over Extract>[AC in #Blue}[AC:Att of AC#Blue ; [in AC:Att#Blue, ID = 1]][Exp in AC#Blue}An American observer who saw [#Red over Extract>[AC in #Red}[AC:Att of AC#Red ; [in AC:Att#Red, ID = 2][Longer:LinkTo, ID = -> 1]][Exp in AC#Red}the beating<#Blue over Extract]{Exp in AC#Blue]{AC in #Blue] that happened then{Exp in AC#Red] reported that the officer "really worked [Exp in AC}him{Exp in AC] over". After [Exp in AC#Red}the beating{Exp in AC#Red]{AC in #Red]<#Red over Extract], [Exp in AC}the prisoner{Exp in AC]{AC in Extract]was forced to remain standing for hours.{Extract]
```

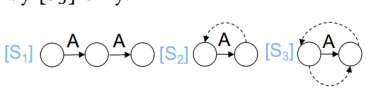
4.3 Parsing LeAG

Let us consider that two eAG are **isomorphic** iff there is a bijective morphism ϕ between them so that a node and its image by ϕ share the same reference value.

LeAG is designed as a markup syntax for eAG. Ideally, there should have been a bijection between LeAGs and the classes of isomorphic eAGs. Yet, this is not the case: first, because two equivalent LeAG documents shall translate into the same eAG, and second, because several non-isomorphic eAGs could match a given LeAG – which is clearly problematic when considering parsing LeAG documents into eAG. For instance, the elementary LeAG `[A}...{A][A}...{A]` may reasonably translate into either of the following:



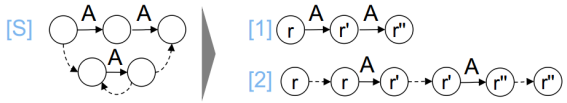
or any eAG made out of a sequence of two edges labelled *A* with the right reference values, separated by any number of epsilon edges. The problem is we cannot, in the absolute, prefer one eAG over the others, since all of them do represent the fact the LeAG document contains two *A* elements in a row – and also, and most importantly, because the different eAG *will not be validated against the same schemas.* Indeed, considering the three SeAGs below, the above eAG [1] is validated by the schema [S_1] only, [2] by both [S_2] and [S_3], and [3] by [S_3] only.



Choosing one solution against the others thus cannot be done but by considering a predefined schema. Hence parsing a LeAG means: given a SeAG, yielding a valid eAG that 'represents well' the LeAG – if such an eAG exists.

In the following, we discuss how to design a deterministic schema-aware LeAG parser. First, we propose to restrict SeAGs to non-ambiguous (N-A) ones [5], ambiguous schemas resulting in non-determinism. We then show that associating to the initial LeAG an eAG, validated by a N-A SeAG, containing the same sequences of elements, whose elements' span is the same, and whose inclusion relations are the same as in the LeAG, is deterministic in general – but not in some cases. We identify those cases and we show that there is a notion of a minimal eAG, that enables to deterministically single out one valid eAG among the others (up to isomorphism[13]).

Non-ambiguous SeAG. A SeAG is non-ambiguous (N-A) iff given any label sequence *w*, there is at most one path connecting the root of the SeAG to its leaf that, epsilon edges set apart, spells *w* [5]. Non N-A SeAG will result in non-deterministic parsing. See the following schema, that matches the LeAG `[A}...{A][A}...{A]` in two different ways – see graphs [1] and [2] below:



LeAG-eAG label sequences. First, we want to stress the fact the sequence of the tags in a LeAG implies the sequences of labels along the different paths the corresponding eAG is made out of, provided the eAG is so that each element of the LeAG has one and only one corresponding element in the eAG.

First let us consider hierarchical LeAGs. The rule 4.2.3 implies there is only one way to read the tags from a given train of tags, regardless of the order in which they are written: this ensures that each of the set of tags of the same colour, in a LeAG, defines one and only one hierarchy of elements. Since a given hierarchy of elements translates into one and only one sequence of labels (epsilon edges set apart) in the eAG model[14], a hierarchical LeAG can be associated only one label sequence in the eAG model – epsilon edges set apart. Now given grafts are hierarchies of elements that are included in a given element of the LeAG[15], and given links and quotes are also hierarchical structures whose connection with the rest of the eAG is determined by the identifier of their root and leaf, the previous discussion extends to LeAGs in general: the sequence of the labels along the paths of the eAG representing a graft, link, quote, is deterministically implied by the LeAG.

eAG equivalence. Two non-isomorphic eAGs are **equivalent** iff their elements form bijective pairs, so that: 1) the elements from each pair share the same name and 2) their previous, following and father and son elements, if they exist, form pairs, and 3) so that the reference values and identifiers at their root/leaf are identical.

Finding a valid eAG. Hierarchies form paths in an eAG. The label sequence of each hierarchical structure of an eAG matching a given LeAG is, as shown above, uniquely defined by the LeAG.

[13] The whole discussion that follows is 'up to isomorphism'.

[14] E.g. '*A* contains *X*' translates into the label sequence: [*X:In* / *A* / *X:Out*].

[15] We consider grafts are *included in* their context element, while an element might belong to both a graft and an outer element: it is always possible to change a graft sharing an element with its context into two grafts strictly included in this context.

Validating a LeAG L is then quite simple. If L is hierarchical, be ls the eAG label sequence corresponding to L. If there is a path in the schema whose label sequence, ϵ edges apart, spells out ls, the LeAG is valid and a valid corresponding eAG can be defined. If L contains one graft: be ls the eAG label sequence matching the uncoloured hierarchy within L, and l' the label sequence for the graft: L is valid if the schema contains a path that spells out ls and another path, inside the element in that path that matches the element in which there is a graft, that spells out l'. Recursively, this principle applies for grafts on grafts; it can be adapted for links and quotes.

Conveniently, in a N-A SeAG, two paths cannot spell the same label sequence, so if there is a path in a schema, say, that matches a hierarchical label sequence, then it is unique. Identically, if, in the context of an element, there is a path that matches the label sequence of a graft, it is also unique. If there are more than one valid eAG, then we know they still all correspond to the same paths in the schema. This means that some schemas are non-deterministic, i.e. that they validate several non-isomorphic, equivalent eAGs.

N-A Schemas validating several equivalent eAGs. We provide the following result: non deterministic N-A SeAGs are the SeAGs that contain a sequence of **at least two epsilon edges**.

A sequence of (two) epsilon edges may happen in two different patterns: the edges either form a cycle or not. The first, general problem with a sequence of two epsilon edges is, since epsilon edges are not represented in LeAG, that the reference values of the nodes of the sequence that do not work as the root or the leaf of an element are undetermined. For instance, the reference of the orange node in I'_{Sa} on figure 6 can be given any value between r_1 and r_2. Hence, several eAGs, differing only by one reference value, can be associated to the left LeAG on that figure.

Additionally, non-determinism may result in structurally different graphs. Cyclic sequences of two epsilon edges may result, in a valid, hierarchical eAG, in any even sequence of epsilon edges. Then, single-layered LeAG annotations, that translate into single-pathed eAGs, will be associated an infinity of equivalent, valid eAGs: see I_{Sa} in figure 6. N-A schemas containing only linear sequences of two epsilon edges will, on the contrary, be deterministic for the parsing of hierarchical LeAGs, but not in case of simulation-based multilayering (figure 6, right). Cyclic epsilon sequences will also be problematic in case of simulation-based multilayering.

One can check that the cases of non-determinism above necessitate a sequence of no less than two epsilon edges: the schemas that contain no or isolate epsilon edges are deterministic.

Minimal eAG. In order to make LeAG parsing with N-A schema deterministic, that is, to ensure that an equivalent class of LeAG documents be associated only isomorphic eAGs, we must single out one of the equivalent eAGs as a unique parsing solution.

Let $\mathcal{X}_S$ be an equivalent class of eAGs validated by the same schema S. For $G \in \mathcal{X}_S$, let us denote $|V_G|$ and $|E_G|$ the cardinal of its sets of nodes and edges respectively. We claim that, up to isomorphism, there is only one eAG $M \in \mathcal{X}_S$ that minimises both the values $size = |V_M| \times |E_M|$ and $Sref = \sum_{v \in V_M} ref(v)$. Then let M be the right representative of that eAG class for parsing under S.

LeAG parsability. For any class of LeAG documents, there is at most one minimal eAG validated by a given N-A SeAG schema, that preserves the LeAG elements, elements' span, identifiers and inclusion relations.

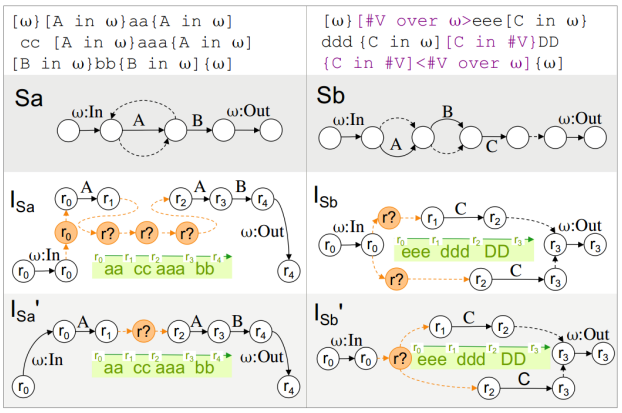


Figure 6: Parsing a LeAG (top, right and left) against a N-A SeAG schema (middle) may yield more than one equivalent eAG, either differing on reference values (r?) or by their structure (dotted orange epsilon edges).

5 RELATED WORK

Many schema-aware data models have been specifically proposed to overcome the limitations of XML by enabling, at the very least, the expression of not only one, but several hierarchies onto the same resources. Among the most notable such 'multistructured' data models, we may mention MulaX [13], XConcur [24], MSXD [7], Rabbit/Duck grammars [27]. Those are all built upon XML and make use of the same fundamental notions like elements, attributes, inclusion, etc. The general validation strategy for those models is to extract or isolate the different hierarchies of elements present in the documents, and validate each separately; they also investigate inter-hierarchy constraints.

LMNL [32] represents a more stripped-down vision of multilayer annotation. In many respects, LeAG borrows from LMNL. In LMNL, the user can identify *ranges* in a character stream and name them by means of pairs of opening and closing tags. Ranges themselves can be annotated by (meta)ranges, which inspired the attribute syntax in LeAG. Yet LMNL claims to be an *annotation* language solely, and not a *structuring* language: in particular, LMNL does not provide the user with means to represent inclusion or sibling relations. As we have seen in paragraph 4.1, indeed, inclusion is either represented by nesting, which limits the data model to trees, or by means of an explicit syntax; LMNL does not propose such syntax, and yet allows overlap and multilayering. By sweeping out the notion of inclusion, LMNL seemingly clears the paradox out; yet LMNL is not absolutely blind to the charms of hierarchies: it lies upon the notion of 'layers', that is, ranges that fully contain the ranges that start and end in their scope, which is reminiscent of XML hierarchies – but if such patterns cannot be interpreted in structural terms, can they be but fortuitous patterns? Still, because hierarchies are a classic and fundamental annotation structure [34], the LMNL model comes along with XML generators that can extract hierarchies from the data. Our point, on that matter, is that since hierarchies are so central, the best is to enable the editors to have direct control over their expression – which indeed demands additional syntax. Apart from those critical considerations, LMNL

is an important annotation model, that goes beyond most others, in terms of expressivity; moreover, it benefits from a grammar-based validation language [31], able to embrace the multilayer documents as a whole, which can be compared only to RDF validators (or to the SeAG we propose [2]).

Indeed, several annotation models have originated from the RDF community. One may think of the pioneering RDFTef [33], the Open Annotation data model [22] or EARMARK [16]. The RDF data model, which imposes no restriction on the shape of the resulting graph, is very expressive; moreover, RDF annotation can be used as a complement to an existing TEI annotation [1], which is a way to ally the best of two worlds. One limitation though of RDF-based annotation languages is the current lack of a proper and computationally efficient validation mechanism. OWL is not natively suitable for validation [21, 25]; tweaks aiming at the expression of, say, integrity constraints, have been experimented, but results in huge execution time [30]. Nonetheless, RDF validation is a promising field of research, as illustrated by the ShEx [18] and SHACL [15] projects. Time complexity still seems to be quite high, but cutting it down is being investigated [29].

6 CONCLUSION

In this paper, we introduced LeAG, an inline markup syntax for extended Annotation Graphs. eAG is a *stand-off* annotation model, based upon a general, cyclic graph formalism: as one may expect, the model is thus highly expressive, fit to express multilayer annotation, but also distant connections across the annotation. The LeAG syntax illustrates that a similarly expressive *inline* markup model can be defined, at least for textual annotation. To achieve that goal, we defined a limited number of necessary syntactical structures (grafts and inserts). This way, the LeAG syntax is kept as simple as possible, while opening wide prospects in terms of editorial enrichments.

We also established the parsability of LeAG into eAG, that is, the fact a LeAG document translates into one corresponding eAG. This aspect of the LeAG syntax is crucial, since it indirectly provides the LeAG documents with a validation language, that is, SeAG, that comes as the validation mechanism for eAG. SeAG does not rely upon the notion of grammars, like most validation languages do, but on the simulation relation. From a technical point of view, simulation-based validation shines by enabling the validation of cyclic data, while keeping the algorithmic costs low. Moreover, as illustrated throughout this article, the ability of SeAG schemas to validate anaphoric chain annotations, which is a canonical linguistic annotation, also evidences the *editorial* relevance of that particular kind of validation for annotation purposes.

REFERENCES

[1] Gioele Barabucci, Angelo Di Iorio, Silvio Peroni, Francesco Poggi, and Fabio Vitali. 2013. Annotations with EARMARK in practice: a fairy tale. In *Proceedings of the 1st International Workshop on Collaborative Annotations in Shared Environment: metadata, vocabularies and techniques in the Digital Humanities*. ACM, 11.

[2] Vincent Barrellon, Pierre-Edouard Portier, Sylvie Calabretto, and Olivier Ferret. 2016. Schema-aware Extended Annotation Graphs. In *Proceedings of the 2016 ACM symposium on Document engineering*. ACM.

[3] Soběslav Benda, Jakub Klímek, and Martin Nečaskỳ. 2013. Using schematron as schema language in conceptual modeling for XML. In *Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling-Volume 143*. Australian Computer Society, Inc., 31–40.

[4] Steven Bird and Mark Liberman. 2001. A formal framework for linguistic annotation. *Speech communication* 33, 1 (2001), 23–60.

[5] Anne Brüggemann-Klein. 1993. Regular expressions into finite automata. *Theoretical Computer Science* 120, 2 (1993), 197–213.

[6] Gerrit Brüning, Katrin Henzel, and Dietmar Pravida. 2013. Multiple encoding in genetic editions: the case of" Faust". *Journal of the TEI* 4 (2013).

[7] Emmanuel Bruno and Elisabeth Murisasco. 2006. MSXD: a model and a schema for concurrent structures defined over the same textual data. In *Database and Expert Systems Applications*. Springer, 172–181.

[8] Hugh A Cayless. 2013. Rebooting TEI Pointers. *Journal of the Text Encoding Initiative* 6 (2013).

[9] Charles Chastain. 1975. Reference and Context. In *Language, Mind, and Knowledge*, Keith Gunderson (Ed.). Vol. 7. University of Minessota Press, Chapter 4, 194–231.

[10] TEI Consortium, Lou Burnard, Syd Bauman, and others. 2008. *TEI P5: Guidelines for electronic text encoding and interchange*. TEI Consortium.

[11] Dan Cristea, Nancy Ide, and Laurent Romary. 2009. Marking-up multiple views of a Text: Discourse and Reference. *arXiv preprint arXiv:0909.2715* (2009).

[12] Paolo D'Iorio and Michele Barbera. 2011. Scholarsource: A digital infrastructure for the humanities. *Switching Codes. Thinking through New Technology in the Humanities and the Arts* (2011), 61–87.

[13] Mirco Hilbert, Andreas Witt, and Oliver Schonefeld. 2005. Making CONCUR work. In *In Extreme Markup Languages*.

[14] HV Jagadish, Laks VS Lakshmanan, Monica Scannapieco, Divesh Srivastava, and Nuwee Wiwatwattana. 2004. Colorful XML: one hierarchy isn't enough. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 251–262.

[15] Holger Knublauch and Arthur Ryman. 2015. Shapes Constraint Language (SHACL). *W3C First Public Working Draft* 8 (2015), W3C.

[16] Silvio Peroni. 2014. Markup beyond the trees. In *Semantic Web Technologies and Legal Scholarly Publishing*. Springer, 45–93.

[17] Pierre-Édouard Portier, Noureddine Chatti, Sylvie Calabretto, Elöd Egyed-Zsigmond, and Jean-Marie Pinon. 2012. Modeling, encoding and querying multi-structured documents. *Information Processing & Management* 48, 5 (2012), 931–955.

[18] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. 2014. Shape expressions: an RDF validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*. ACM, 32–40.

[19] Francesco Ranzato and Francesco Tapparo. 2010. An efficient simulation algorithm based on abstract interpretation. *Information and Computation* 208, 1 (2010), 1–22.

[20] Allen H Renear, Elli Mylonas, and David Durand. 1993. Refining our notion of what text really is: The problem of overlapping hierarchies. (1993).

[21] Dave Reynolds, Carol Thompson, Jishnu Mukerji, and Derek Coleman. 2005. An assessment of RDF/OWL modelling. *Digital Media Systems Laboratory, HP Laboratories Bristol* 28 (2005).

[22] Robert et al. Sanderson. 2013. Open annotation data model. *W3C community draft* (2013).

[23] Desmond Schmidt. 2012. The role of markup in the digital humanities. *Historical Social Research/Historische Sozialforschung* (2012), 125–146.

[24] Oliver Schonefeld. 2007. XCONCUR and XCONCUR-CL: A constraint-based approach for the validation of concurrent markup. In *Data Structures for Linguistic Resources and Applications. Proceedings of the Biennial GLDV Conference*.

[25] Evren Sirin. 2010. Data validation with OWL integrity constraints. In *Web Reasoning and Rule Systems*. Springer, 18–22.

[26] C Michael Sperberg-McQueen. 1991. Text in the electronic age: Texual study and textual study and text encoding, with examples from medieval texts. *Literary and Linguistic Computing* 6, 1 (1991), 34–46.

[27] C Michael Sperberg-McQueen. 2006. Rabbit/duck grammars: a validation method for overlapping structures. In *Extreme Markup Languages*.

[28] C Michael Sperberg-McQueen and Claus Huitfeldt. 2000. Goddag: A data structure for overlapping hierarchies. In *Digital documents: Systems and principles*. Springer, 139–160.

[29] Slawek Staworko, Iovka Boneva, Jose E Labra Gayo, Samuel Hym, Eric G Prud'hommeaux, and Harold Solbrig. 2015. Complexity and Expressiveness of ShEx for RDF. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[30] Jiao Tao, Evren Sirin, Jie Bao, and Deborah L McGuinness. 2010. Integrity Constraints in OWL.. In *AAAI*.

[31] Jeni Tennison. 2007. Creole: Validating overlapping markup. In *Proceedings of XTech*.

[32] Jeni Tennison and Wendell Piez. 2002. The Layered Markup and Annotation Language (LMNL).. In *Extreme Markup Languages*.

[33] Giovanni Tummarello, Christian Morbidoni, and Elena Pierazzo. 2005. Toward Textual Encoding Based on RDF.. In *ELPUB*.

[34] Andreas Witt. 2010. Different views on markup. *Text, Speech and Language Technology* (2010), 1.