Schema-aware Extended Annotation Graphs

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

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

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

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

ABSTRACT

Multistructured (M-S) documents were introduced as an answer to the need of ever more expressive data models for scholarly annotation, as experienced in the frame of Digital Humanities. Many proposals go beyond XML, that is the gold standard for annotation, and allow the expression of multilevel, concurrent annotation. However, most of them lack support for algorithmic tasks like validation and querying, despite those being central in most of their application contexts.

In this paper, we focus on two aspects of annotation: data model expressiveness and validation. We introduce extended Annotation Graphs (eAG), a highly expressive graph-based data model, fit for the enrichment of multimedia resources. Regarding validation of M-S documents, we identify algorithmic complexity as a limiting factor. We advocate that this limitation may be bypassed provided validation can be checked *by construction*, that is by constraining the shape of data during its very manufacture. So far as we know, no existing validation mechanism for graph-structured data meets this goal. We define here such a mechanism, based on the simulation relation, somehow following a track initiated in Dataguides. We prove that thanks to this mechanism, the validity of M-S data regarding a given schema can be guaranteed without any algorithmic check.

Keywords

Multistructured data model; Validation; Schemas; Graphs.

1. INTRODUCTION

Multistructured (M-S) data models have been a hot topic for over a decade. Correlated to the rise of Digital Humanities, they ground on the fact that a single hierarchy is not always sufficient to represent annotated resources [6], contrasting with the setting of XML-based languages as a standard for scholarly annotations.

DocEng 2016 Vienna, Austria

ISBN 978-1-4503-4438-8/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2960811.2960816

Hence, "multistructured" is then to be understood by comparison with XML: annotating somehow means *structuring* data (a well-formed XML document fits into a tree *structure*); "multi" suggests M-S data models handle multiple, interlaced hierarchical annotations over the same data. Many models have been proposed [18]. However, the enhanced expressiveness resulting from less constrained structural foundations, compared to XML, comes at a cost: M-S models often lack support for tasks like querying or validation which are commonplace in XML.

Indeed, validating highly expressive data is challenging, due to a general trade-off between data models expressiveness and algorithmic complexity. The NEXPTIME complexity of OWL/DL inference [20] will serve as a striking example of how costly the validation of highly expressive graph-structured documents can be. This trade-off is so pregnant and restrictive that it applies to XML [23]. Hence the need for alternative validation strategies for M-S data.

We introduce here "simulation-based validation by construction" for M-S data. Our main results are the following:

- We designed eAG, an expressive M-S data model based on Annotation Graphs (AGs) [2]. We strengthened the AG model in order to address its main deficiencies, like a shared representation for inclusion and cooccurrence and a limited ability to handle composite resources.

- We identified the *simulation* relation, first used for the structural description of semistructured data [5], as a promising mechanism for eAG validation. We designed SeAG, a simulation-based schema model for eAG.

- We defined a coupled representation for SeAGs and eAGs so that, given the representation of a schema, only valid eAGs can be represented: this is "validation by construction". This enables to guarantee the validity of rich M-S data *without algorithmic check*, bypassing the trade-off between expressiveness and complexity, when schema definition can precede annotation.

- We found that the eAG/SeAG model is compatible with classical, *a posteriori* validation. In this case, checking whether an eAG is valid against any schema can be decided in polynomial time ($O(|edges| \cdot |nodes|)$).

- We finally proved that for hierarchical data, SeAG syntactic validation is not less straitening than Relax-NG.

"Validation by construction" can be seen as a special *on-the-fly* validation mechanism. First, we prove the interest of such validation in a (not exclusive) application context. A panorama of M-S models and validation mechanisms follows. Eventually, we formally introduce eAGs and SeAGs.

2. APPLICATION CONTEXT

Among other application fields of M-S data, eAG/SeAG are particularly fit for scholarly digital publishing projects. Four such projects are associated to this work, dealing with Diderot's *Encyclopédie*, Stendhal's *Journaux et Papiers*, Desanti's archive and Flaubert's documentation for *Bouvard and Pécuchet*[1]. Such projects may benefit from a M-S data model supporting on-the-fly validation for two reasons.

Data model expressiveness. Editors are mainly expert humanists; their aim is to express complex and accurate information about the corpus they edit through annotation. They may want to annotate data according to several competing paradigms, resulting in non-hierarchical annotations[2]. For economical reasons, not all editorial projects benefit from technical support, which means editors often face annotation encoding alone. Thus, there is a potential discrepancy between the technicality of annotation and the technical skills of the editors. One (formally elegant) way to bypass this discrepancy is to provide them with very expressive M-S data models, so that non-hierarchical information encoding, that is tricky in XML, becomes straightforward.

Editorial routine. Editorial routines starts with the definition of the *editorial policy*, which sets the nature of the critical enrichments. In XML-based digital publishing projects, *schemas* are a common way to represent this policy. Transcribing the editorial principles into a schema guarantees a certain harmony in annotation – hence the need for a schema-aware M-S data model. Moreover, most XML tools provide the user with content assist (i.e. on-the-fly validation) features, acting as an authoring tool that suggests elements according to the editing context. This feature, that helps commitment into deep annotation, is valuable in a publishing context, where schemas are defined prior to annotation. This is worth translating into the M-S world.

3. RELATED WORKS

XML-TEI is the standard for scholarly publishing. It provides scholars with a modular, versatile and documented schema, and user-friendly XML editors are plenty. Moreover, XML is a natural candidate for annotation. Annotation models need to support: 1. linear characterization (e.g. along *the* reading dimension – if unique), 2. representation of inclusion (e.g. to encode *the* material structure of a text – if unique), 3. of disseminated elements and links. XML does the first two well (in case of uniqueness, above): elements are ordered along the text; nesting represents inclusion.

Still, XML does not suit some common annotation patterns [6]. Overlapping elements are not allowed; links do not have a syntactical representation, so they need separate validation (e.g. with Schematron); non inclusive nesting, frequent in case of multiple annotation, cannot be represented. Some works aim at conforming TEI-XML with more expressive data models [6, 7], but they either fail to tackle some of XML inherent limitations (e.g. *nesting* representing *inclusion*) or lose compliance with XML tools (XSD, XQuery, etc.) [18].

[1]The project's Websites are: enccre.academie-sciences.fr ; manuscrits-de-stendhal.org ; archive.desanti.huma-num.fr ; dossiers-flaubert.fr

[2]See [6], ch. 20.

Multistructured data models.

Formally speaking, annotated resources can be regarded as labelled graphs [5]. From there on, the expressive limitations above appear as a consequence of the overly restricted family of graphs upon which XML is based: trees. This formal reasoning gave birth to Competing markup or Multistructured (M-S) data models [18]. The term "multistructured" refers to what *structure* means in XML: in a M-S data model, well-formedness extends from trees to (at least) forests, graphs whose connected subgraphs are trees.

CONCUR [9] precisely enhances SGML to support *forests* of elements. Each tree is defined in a DTD; inside a CONCUR document, the tags explicitly relate to the tree they belong to. MuLaX [11] transposes this philosophy to XML, despite XML documents referring to at most one schema. A MuLaX document is a mix of overlapping elements from disjoint hierarchies; each hierarchy defines a projection, yielding a well-formed XML document that can be validated against a schema. Those solutions, however, do not support cross-hierarchies constraints; self-overlap is also problematic[3]. MSXD [3] implements such constraints. Its formal model is a forest, but unlike the above solutions, the different trees are instantiated in distinct documents. RelaxNG schemas validate each. The relative position of elements from different trees can be constrained by Allen's relations.

Other models rely on well identified and wider graph families like multitrees, where trees may share nodes (TexMecs [12]), multicolored trees (MCT, [13]) or restrained, acyclic polyarchies (GODDAG [22]). Multitrees can be handled by an ingenious grammar-based validation language, dubbed Rabbit/duck Grammar [21]. While checking some crosswise constraints, a Rabbit/duck Grammar extracts hierarchies from the multitree structured document; the extracts are then validated against XML schemas. Rabbit/duck Grammars manage self-overlap.

More expressive standoff, graph-based models, exemplified by LMNL [27], have been proposed. A LMNL document is a layered directed acyclic graph where elements are labelled ranges from a character stream. Since they are text streams themselves, even annotations can be annotated. Creole, a powerful grammar-based schema language, validates LMNL [26]. Annotation Graphs [2] is another model based upon a handy notion of chronology for multimedia corpus annotation. A few RDF-based annotation models have also been proposed, amongst which EARMARK [17], which is built against a dedicated OWL ontology that adds semantics to the logical, directed cyclic graph formalism that RDF is, stands out.

M-S validation : algorithmic complexity.

As detailed above, most M-S validation mechanisms proceed tree after tree, providing the final user with a clumsy modelling tool. MSXD only enables to express weak constraints betweens trees. RdGs, which somehow manages to embrace multitrees, fall short when it comes to validating more general graphs. An explanation to that glass ceiling might be found in time complexity[4]. In general, the more expressive the data model, the higher the complexity for the related processing tasks [23, 15, 14, 20]. This applies, at least, to grammar-based and rule-based validation.

The three main validation languages for XML, namely DTD, W3C XML Schema (XSD) and RelaxNG, are commonly modelled as tree grammars, or tree automaton [15]. Although not equal, the languages that those schemas recognise fit into regular tree lan-

[3]Since individual schemas cannot define overlap, an arbitrary number of schemas is required to validate multiple, self-overlapping elements [26]. This also applies to MSXD.

[4]For validation, time- prevails over space-complexity [15].

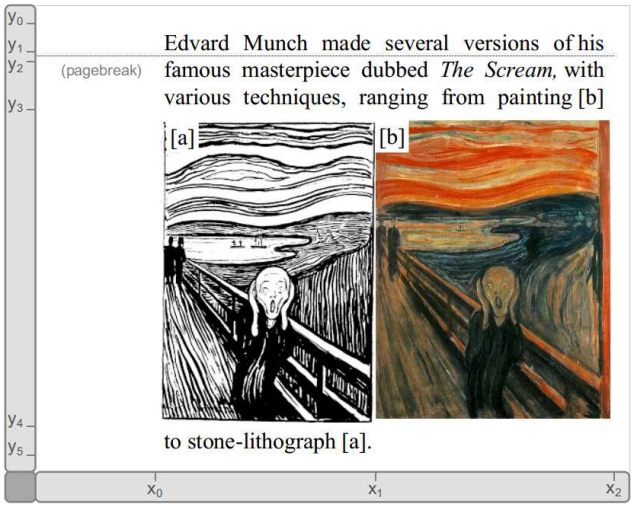


Figure 1: Document showing overlap, a figure enclosed in text and internal references.

guages[5], for which validation can be done in linear time in the documents' size [14]. Other tree languages may not even be decidable for interpretation [23], which is part of validation in RelaxNG and XSD (see Figure 5).

Little information is available about the complexity of the advanced M-S validation mechanisms. The processing cost of the initial MSXD schema mechanism, combining RelaxNG with Allen's relations, was left undetermined [3]. In the end, it seems not to have been implemented as such [4]: XQuery extensions were designed as surrogates for Allen's relations and cross-hierarchies constraints rather checked by querying the data. Creole, the only consistent schema mechanism validating documents with layered, overlapping annotations, was prototyped using XSLT; despite RelaxNG-inspired optimization, the result was considered "too slow" [26] – no other implementation followed, unfortunately.

RDF-based M-S data models suffer from the same trade-off. OWL reasoners are sometimes used as validators [8, 17]. However, using OWL rules to validate a document is problematic. First, OWL-Full is undecidable and the two main restrictions, OWL-DL and OWL-Lite, perform in NEXPTIME and EXPTIME respectively [20]. Second, OWL rules are not natively interpreted as integrity constraints[6], resulting in a weak validation mechanism [20]. Experimental techniques to by-pass those limitations seem to result in huge execution times [25].

4. PROPOSITION

The above enlightens a trade-off between expressiveness and complexity – trade-off that expresses, to be accurate, *inside the frame of a given validation technique*. For instance, while Brzozowski derivative-based validation [26] runs in linear time for regular tree languages, the same approach does not extend easily to more general graphs. This leads to question the use and tweak of XML and RDF tools for M-S validation, precisely because, as well engineered systems, they are already optimized for their native use.

Simulation [19, 5], is an interesting alternative to rule- and grammar-based descriptive formalisms. A simulation is a relation over (often rooted) directed labelled graphs. Informally, the existence of a rooted simulation of a graph B by a graph A implies that all the paths of B starting from its root have a matching path in A, whose label sequence is identical. Thus, A describes the structure of B, because all the patterns in B somehow have a match in A. Conversely, A behaves as a graph schema: it validates the graphs that contain only patterns defined in A, i.e. that A simulates.

Validation by simulation was first operated for semistructured (S-S) data [24, 1]. The Object Exchange Model (OEM) underlying S-S data is a cyclic, unordered, directed labelled graph. Natively, a S-S database is schemaless; Dataguides [10] or Graph Schemas [5] are inferred from the data. They are graphs that simulate the S-S database, providing a structural description that can be exploited for querying purposes. Simulation check performs in $O(|edges| \cdot |vertices|)$ [19], which is acceptably low for general graph-structured data validation.

Still, despite providing an expressive data model[7] and an appropriate schema mechanism, as far as we know, S-S model was never tuned for annotation. Indeed, it lacks a clear representation of inclusion, a notion of order or a way to index nodes along reading dimensions to support linear annotation; moreover, since Dataguides and Graph Schemas are inferred from the data, they cannot be used as authoring tools.

Still, because the OEM is so general, the principle of a simulation-based validation is not restricted to S-S data but "can be applied easily to any graph-based data model" [10]. We propose here a data model, fine-tuned to comply with simulation-based validation, as we will elaborate.

4.1 Data model: eAG

The data model we propose is extrapolated from Annotation Graphs (AG) [2], hence named extended AG (eAG). It is a standoff markup formalism. The toy document represented in figure 1 will serve to illustrate eAG's expressive power. It is made of one paragraph spanning over two pages, whose text locally refers to parts of a figure. The figure itself, *accidentally*, nests inside the paragraph (without being *part* of it).

An eAG $G = (V, E)$ is a directed, connected and labelled cyclic graph, with edges E and vertices V. It has only one root and one leaf, denoted $rt(G)$ and $lf(G)$ respectively. It verifies all the properties that follow.

Notation. In the following, $v\lfloor e \rfloor v'$ denotes the graph made out of the edge e connecting the node v to v'. $label(e)$ yields the value of the label of e.

First, we define chronologies, that is how locations in composite resources can be made reference to.

Definition 1. A (general) chronology is any ordered set $\langle T, \leq \rangle$. Be then $\mathscr{C}$ a set of strings called "chronometer names". Be $m \in \mathscr{C}$. The reference space associated to m is a unique ordered set $\langle \mathbb{T}_m, \leq_m \rangle$. A chronology over $m \in \mathscr{C}$ is an ordered set $\langle T, \leq_m \rangle$ so that $T \subseteq \mathbb{T}_m$.

Definition 2. (Concatenation) Be $\langle T_a, \leq_a \rangle$ and $\langle T_b, \leq_b \rangle$ chronologies. $\langle T_a \cdot T_b, \leq_{a,b} \rangle$ defines a chronology over $T_a \cup T_b$ iff the following relation $\leq_{a,b}$ defines an order over $T_a \cup T_b$:
For any $t, t' \in T_a \cup T_b$, then:
- $t =_{a,b} t' \Leftrightarrow \exists x \in \mathscr{C} \mid (t,t') \in \mathbb{T}_x^2 \wedge t =_x t'$
- $t <_{a,b} t' \Leftrightarrow \exists x \in \mathscr{C} \mid (t,t') \in \mathbb{T}_x^2 \wedge t \leq_x t'$
  or $\nexists x \in \mathscr{C} \mid (t,t') \in \mathbb{T}_x^2 \wedge (t,t') \in T_a \times T_b$.

[5]Local tree languages (DTDs) restrict single-typed tree languages (XSD), restricting regular tree languages (RelaxNG).

[6]The Open World Assumption and the No Unique Name feature together allow to assess an assertion is verified, but not that is is not.

[7]Surprisingly, the OEM is referred to as "essentially equivalent to XML" on the Lore project website (infolab.stanford.edu/lore), which was a pioneering OEM DBMS before migrating to XML.

Table 1: Allowed suffixes per label class and the resulting class. "-" stands for "undefined".

$\looparrowright$ suffixed by	$\varnothing$	:In	:Out	:Att	:LinkTo
$l \in \mathscr{L}_\varnothing$	$\mathscr{L}_\varnothing$	$\mathscr{L}_{In}$	$\mathscr{L}_{Out}$	$\mathscr{L}_{Att}$	$\mathscr{L}_{LinkTo}$
$l \in \mathscr{L}_{In}$	$\mathscr{L}_{In}$	-	-	-	-
$l \in \mathscr{L}_{Out}$	$\mathscr{L}_{Out}$	-	-	-	-
$l \in \mathscr{L}_{Att}$	$\mathscr{L}_{Att}$	$\mathscr{L}_{In}$	$\mathscr{L}_{Out}$	-	-
$l \in \mathscr{L}_{LinkTo}$	$\mathscr{L}_{LinkTo}$	-	-	-	-

Example 1. Be $T_1 = \{0,1,2\}$ and $\leq_1$ the order on naturals. Be $T_2 = \{X,Y\}$ and $\leq_2$ the lexical order. $\langle T_1 \cdot T_2, \leq_{1,2}\rangle$ defines a chronology, where $2 <_{1,2} b$ for instance. Now, given $T_3 = \{2\}$ and $T_4 = \{4\}$ equipped with the natural order, $\langle T_1 \cdot T_2 \cdot T_3, \leq_{1,2,3}\rangle$ does not define a chronology (the antisymmetry would not hold), while $\langle T_1 \cdot T_2 \cdot T_4, \leq_{1,2,4}\rangle$ does.

Definition 3. (Inclusion) Be $\langle T_a, \leq_a\rangle$ and $\langle T_b, \leq_b\rangle$ chronologies. We say $\langle T_b, \leq_b\rangle \subseteq \langle T_a, \leq_a\rangle$ iff $\exists (T_1, T_2) \subset T_a^2$ so that $\langle T_1 \cdot T_b \cdot T_2, \leq_{a,b,a}\rangle$ defines a chronology.

Example 2. In Example 1, $\langle T_2, \leq_2\rangle \subseteq \langle T_1 \cdot T_4, \leq_{1,4}\rangle$.

This notion of chronology enables to index a composite, yet continuous content.

Illustration, part 1. Consider the second page in Figure 1. It contains three modules, respectively containing two text lines, a figure, and one line. Be the three chronologies: $\langle T_y, \leq_y\rangle$ based on a vertical descending dimension that is not continuous over page changes, with $T_y = \{y_2, y_3, y_4, y_5\}$ (in ascending order), for the three modules delimitation; $\langle T_x, \leq_x\rangle$ based on an horizontal left-to-right dimension, with $T_x = \{x_0, x_1, x_2\}$, for the figure decomposition into images; $\langle T_c, \leq_c\rangle$, with $T_c = \{41, 83, 84, 129, 130, 154\}$ [8], based on characters (including linebreaks) count, for lines indexation. By double inclusion, we can define a chronology $\langle T, \leq\rangle$ over $T_c \cup T_x \cup T_y$, so that $y_2 < 41 < 83 < 84 < 129 < y_3 < x_0 < x_1 < x_2 < y_4 < 130 < 154 < y_5$.

Definition 4. (References) In an eAG, a reference $ref(v)$ is associated to each node v. For each v, there is a unique reference space $\langle \mathbb{T}_c, \leq\rangle_c$ so that $ref(v) \in \mathbb{T}_c$.

Two references belonging to the same chronology and sharing the same reference space identify a range within the resources to be annotated. Ranges can be annotated by creating two nodes bearing the corresponding references, connected by (at least) a directed, labelled edge. In this simple case, the label constitutes the content of the annotation.

To structure further annotation, we define a label semantics to indicate that an annotation is an element, a link, or is included within another, or is an attribute of another.

Definition 5. (Labels) Be a special character ε [9]. Be $\mathscr{L}_\varnothing$, $\varepsilon \notin \mathscr{L}_\varnothing$, a set of strings that do not contain the character ":". $\mathscr{L}_0 = \mathscr{L}_\varnothing \cup \{\varepsilon\}$ is the set of unsuffixed labels. Additionally, be $\mathscr{S} = \{$:In, :Out, :Att, :LinkTo$\}$ the set of suffixes. Labels can be iteratively suffixed according to the rules given in table 1. Those rules also define classes of labels, e.g. $\mathscr{L}_{In}$ the set of labels whose last suffix is :In. The set of all labels $\mathscr{L}$ is the union of all the preceding classes.

[8] The first line of page 2 starts at character 41, etc.

[9] ε stands for a blank, or void, annotation.

In the following, given two strings l and s, $s \subset l$ denotes the fact that l contains the substring s.

One asset of eAG is a clear distinction between accidental nesting and inclusion representation. We define here inclusion, based upon the :In and :Out suffixes.

Definition 6. (h-equality and dominance). Be $G = (V,E)$ a graph, and $(\{v_0 ... v_N\}, \{e_0 ... e_{N-1}\})$ an h-path of G. Be $n, m \in \mathbb{N}$; $0 \leq n \leq m \leq N$.
v_n is said to be h-equal to v_m, denoted $v_n =^h v_m$, iff :
1. $n = m$ or
2. $\forall j \in [n, m-1]$, $label(e_j) \in \mathscr{L}_0 \cup \mathscr{L}_{Att}$ or
3. $v_n \not>^h v_m \ \wedge \ v_m \not>^h v_n$ and $\forall k, l \ ; \ n < k \leq l < m$, $v_n \geq^h v_k \wedge v_n \geq^h v_l \wedge v_m \geq^h v_k \wedge v_m \geq^h v_l$ and $v_k =^h v_l \vee (v_k >^h v_l \ \vee \ v_l >^h v_k)$ [see right below].
When $n \neq m$, v_n and v_m are said to border-h-dominate the nodes $v_i, i \in [n+1, m-1]$, denoted $(v_n, v_m) >_b^h v_i$, iff :
1. $\exists l \in \mathscr{L}$; $label(e_n) = l$:In and $label(e_{m-1}) = l$:Out and
2. $\forall j < k \in [n+1, m-1], v_j =^h v_k$
or $\exists (l,m) \in [j, k-1] \times [k+1, m-1]; (v_l, v_m) >_b^h v_k \wedge v_j =^h v_l$
or $\exists (l,m) \in [n+1, j-1] \times [j+1, k]; (v_l, v_m) >_b^h v_j \wedge v_k =^h v_m$.
Be $x, y \in [0, N]$. v_x h-dominates v_y, denoted $v_x >^h v_y$, iff $\exists n, m \in \mathbb{N}$; $0 \leq n \leq m \leq N \mid (v_n, v_m) >_b^h v_y \wedge v_x =^h v_n$.

Property 1. $\forall v \lfloor e \rfloor v' \subseteq G$, $v \neq^h v' \ \wedge \ v \not>^h v' \ \wedge \ v' \not>^h v$ is equivalent to "$:LinkTo$" $\subset label(e)$.

Property 2. Be $G = (V,E)$. $\forall v \lfloor e \rfloor v' \subseteq G$ we enforce that: $label(e) \in \mathscr{L}_{In} \Leftrightarrow (v >^h v')$ and $label(e) \in \mathscr{L}_{Out} \Leftrightarrow (v' >^h v)$

This means that an edge labelled l:In does not go without a h-dominated path ending by an edge labelled l:Out.

Definition 7. (h-levels) Be an eAG G. Since G is connected, $\forall v \in V, \exists P = (V_P, E_P) \subseteq G$ a root-to-leaf path so that $v \in V_P$. The h-level of v in P is the biggest subset $N \subseteq V_P$ so that $\forall v' \in N, v' =^h v$.
(h-levels direct inclusion) Be a path P, N_x, N_y h-levels in P. N_y is directly included in N_x, denoted $N_x \sqsupset^h N_y$, iff :
1. $\forall (v_x, v_y) \in N_x \times N_y, v_x >^h v_y$ and
2. $\nexists v \in V \mid v_x >^h v >^h v_y$.
(h-levels inclusion) The h-inclusion is the transitive closure of $\sqsubset^h$. It is denoted $\subset^h$.
($\mathscr{P}r$ and $\mathscr{S}c$) An h-level N is primary (secondary), denoted $N \in \mathscr{P}r$ (resp. $N \in \mathscr{S}c$) iff $\forall N' \subseteq^h N, \forall (v, v') \in N \times N'$, if $\exists e$ so that $v' \lfloor e \rfloor v \subseteq G \ \vee \ v \lfloor e \rfloor v' \subseteq G$, then"$:LinkTo$" $\notin label(e) \Rightarrow$"$:Att$" $\notin label(e)$ (resp. "$:Att$" $\subset label(e)$).

Property 3. Be $G = (V,E)$. We enforce that:
1. For all N h-level of G, $N \in \mathscr{P}r \cup \mathscr{S}c$.
2. For all $N \in \mathscr{S}c, \exists N' \in \mathscr{P}r; N' \sqsupset^h N$.

The above provides us with a definition of inclusion, on which to found a definition of elements and attributes.

Definition 8. (Element, attribute) Be $G = (V,E)$ an eAG. An element (resp. attribute) is a subgraph H of G so that:
1. $H = v \lfloor e \rfloor v' \mid label(e) \in \mathscr{L}_\varnothing$ (resp. $\mathscr{L}_{Att}$) or
2. $H = v_1 \lfloor e_1 \cdots e_{M-1} \rfloor v_M$ so that :
   a. $\forall i \in [1, M-1], (v_1, v_M) >_b^h v_i$ and
   b. $\exists N_H \in \mathscr{P}r$ (resp. $\mathscr{S}c$) ; $v_1, v_M \in N_H$ and
   c. $\nexists H_1 = v_1^1 \lfloor e_1^1 \cdots e_{M_1-1}^1 \rfloor v_{M_1}^1$, $H_2 = v_1^2 \lfloor e_1^2 \cdots e_{M_2-1}^2 \rfloor v_{M_2}^2$ verifying the above conditions a. and b., with $N_{H_1} = N_{H_2}$ and $e_1^1 = e_1 \neq e_1^2 \ \wedge \ e_{M_2-1}^2 = e_{M-1} \neq e_{M_1-1}^1$.

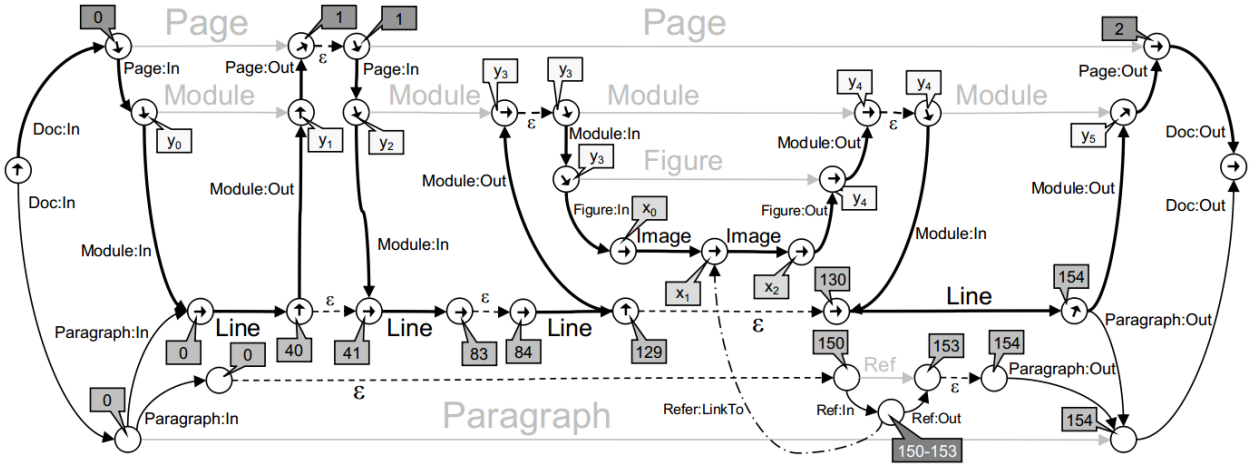


Figure 2: An eAG representing Figure 1 and a way to browse through it (arrows). Grey edges are for reading assistance (they span over the paths defining a structured element). Speech balloons show reference values (shades differenciate between chronometers), from a chronology extending $\langle T, \leq \rangle$ (cf.*Illustration, part 1*) in order to detail the content of Page one and a Ref ("[a]" in the text) between characters 150 and 153.

This defines consecutive elements: two elements A and B are consecutive if, say, $root(B) = leaf(A)$. We can extend that notion to A and B only separated by a series of ε edges. Elements can also include one another, based on the previous definition. We go further and add the following property.

Property 4. Be $A \neq B$ two elements. We enforce that $(lf(A), rt(A)) >_b^h rt(B) \Leftrightarrow (lf(A), rt(A)) >_b^h lf(B)$.

It means that two elements whose roots and leaves are either on the same h-level or on h-levels included one into the other either are *consecutive* (directly or not) or *include* one another. Paths connecting the root of an eAG to its node and made only out of consecutive and inclusive elements will be referred to as "linear annotation paths". An eAG contains several such paths which, individually, represent a given annotation paradigm *à la* XML, since they can be modelled as ordered trees of elements. However, in an eAG, some elements can very well appear simultaneously on several such paths (c.f. *Illustration, part 2*). This means element hierarchies share items: an eAG can be modelled by no less than a multitree. Eventually, because edges whose label contains "`:LinkTo`" are unrestricted, they can connect any nodes together, which may result in a cyclic graph.

Illustration, part 2. (Linear annotation paths) Figure 2 shows an eAG representing the document illustrated in Figure 1. It contains three competing linear annotation paths. The arrowed path provides a layout-oriented Page description, fragmented into Modules, Lines, Figures and Images. Another identifies a Ref inside the text of the Paragraph. The last path splits Paragraphs into Lines. Lines are *shared elements* with the first path; they are also *the only* shared elements. For instance, the Paragraph *does not* include the Figure element, since there is no h-inclusion between the h-levels where the roots and leaves of the two elements appear.
(Structured element, Link) The Ref element is made out of two edges labelled `Ref:In` and `Ref:Out`. It annotates the string "[a]" from inside the text. Graph-wise, it is a structured element, since it contains more than one edge. It is also void, as there is but a node between its two constituting edges; still, this node points towards the second Image on another annotation path by means of an edge suffixed `:LinkTo`.

Property 5. (Covering chronologies) We enforce that:
1. Be an h-level N. $\exists \langle T, \leq \rangle$ a chronology, $\exists! c \in \mathscr{C}$ so that $\forall v \in N, ref(v) \in T \cap \mathbb{T}_c$. This defines a sub-chronology $\langle T_N, \leq_N \rangle$ so that $T_N = T \cap \mathbb{T}_c$ and $\leq_N = \leq_c$.
2. Be $(N, N') \in \mathscr{P}r^2$. $N' \sqsubset N \Rightarrow \langle T_{N'}, \leq_{N'} \rangle \subseteq \langle T_N, \leq_N \rangle$.
3. Be $v \lfloor e \rfloor v'$ so that $\exists (N, N') \in \mathscr{P}r^2; (v, v') \in N \times N'$ and `:LinkTo` $\not\subset label(e)$. Point 1. or 2. (depending on $N = N'$ or $N' \neq N$) ensure that there is a chronology $\langle T, \leq \rangle$ so that $ref(v), ref(v') \in T$. Then $ref(v) \leq ref(v')$.

Linear annotation only makes sense provided there is a dimension along which the elements flow: in particular, Property 5 means that the structural order in which elements are positioned along a linear annotation path must not contradict the order of the references of their nodes. As a consequence, there is always a covering chronology for a linear annotation path. However, it is possible to annotate a resource without, or against, any chronological order, thanks to the unconstrained edges whose label contains `:LinkTo`.

Definition 9. (Accidental nesting) Be an eAG G and A, B two elements. B is accidentally nested in A iff there are:
- two linear annotation paths $P_1 = (V_1, E_1), P_2 = (V_2, E_2)$, their covering chronologies $\langle T_1, \leq_1 \rangle$, $\langle T_2, \leq_2 \rangle$ and $N_A \subseteq V_1$, $N_B \subseteq V_2$ the h-levels (in P_1 and P_2 resp.) so that $root(A), leaf(A) \in N_A$ and $root(B), leaf(B) \in N_B$, and
- $c \in \mathscr{C}, \exists N, N'; N \subseteq^h N_A, N' \supseteq^h N_B$ verifying $N \subseteq V_1, N' \subseteq V_2$, and $\exists (v_\chi, v_\phi) \in N^2, (v_x, v_y) \in N'^2$, so that:
1. $ref(v_\chi), ref(v_\phi), ref(v_x), ref(v_y) \in \mathbb{T}_c$
2. $ref(root(A)) \leq_1 ref(v_\chi)$, $ref(v_\phi) \leq_1 ref(leaf(A))$
3. $ref(v_x) \leq_2 ref(root(B))$, $ref(leaf(B)) \leq_2 ref(v_y)$
4. $ref(v_\chi) <_c ref(v_x) <_c ref(v_y) <_c ref(v_\phi)$.

(Overlap) A and B overlap (with A first) iff the above paths, h-levels, chronometer and nodes exist and verify:
1. $ref(v_\chi), ref(v_\phi), ref(v_x), ref(v_y) \in \mathbb{T}_c$
2. $ref(root(A)) \leq_1 ref(v_\chi) \leq_c ref(v_x) \leq_2 ref(root(B))$
3. $ref(leaf(A)) \leq_1 ref(v_\phi) \leq_c ref(v_y) \leq_2 ref(leaf(B))$.

Example 3. In Figure 2, since the elements Figure and Paragraph are not on the same linear annotation paths, they cannot include one another. However, Figure is accidentally nesting inside the Paragraph.

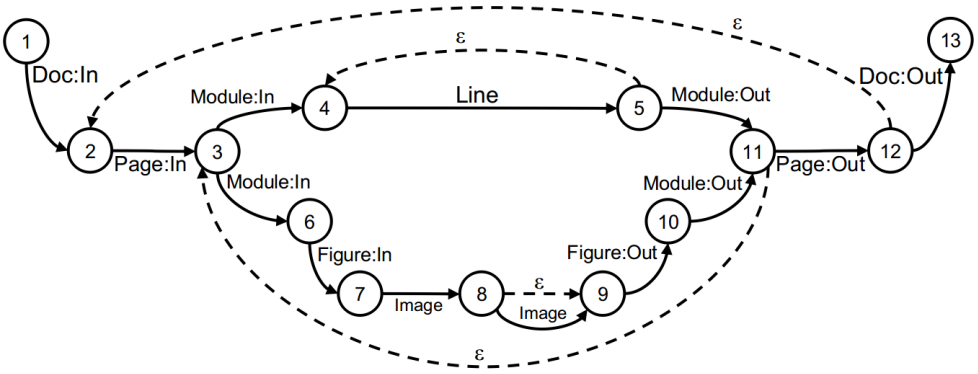


Figure 3: A schema for eAG. It validates the graph from Figure 2, restricted to the arrowed path.

Illustration, part 3. (Linear annotation paths) One can extend the composite chronology $\langle T, \leq \rangle$ defined in *Illustration, part* 1 to cover the whole arrowed path in Figure 2, so that for any node v preceding a node v' along this path, $ref(v) \leq ref(v')$.
(Inter-chronometers comparisons) Cross-chronometer assessments can be made on an eAG. First example, because in $\langle T, \leq \rangle, x_2 < 130$, we know that from a (top-down) layout point of view, the second image precedes the last Line. (Cross-linear paths comparison) Cross-linear annotation path assessments can also be made, thanks to the notions of accidental nesting and overlap. E.g. Ref is accidentally nested in the last Module, because this Module h-includes the Line delimited by characters 130 and 154, while Ref ranges from character 150 to 153. Then, it is possible to assess that the Ref is located further than the last Image, from a descending layout point of view. The edge labelled `Refer:LinkTo` (which is a link) does not respect the inferred reference order, which is not contradictory with the eAG model.

4.2 Schema model

To sum up, an extended Annotation Graph is a connected, directed and labelled graph whose nodes bear references values. Informally, an eAG is composed of several linear annotation paths sharing items and connected together by `:LinkTo` edges.

We have defined linear annotation paths by some specific properties. Those properties, together with the graph model, define what a *well-formed* eAG is.

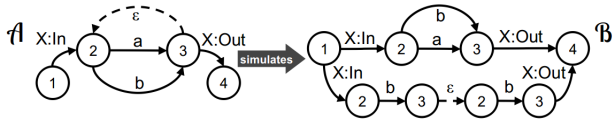
Now we define a schema model for eAG. Schemas are a means to define the allowed elements/attributes and their mutual relationships (consecutiveness, inclusion, existence of `:LinkTo` connexions) for the matching eAGs. Since elements, attributes and relationships have a homogeneous edge-based representation, eAG schemas needs be no more than a *graph description* formalism, which simulation is [5].

Definition 10. (SeAG) An eAG schema, denoted SeAG, is a directed, connected, labelled graph with one root and one leaf only. Its labels fall into Definition 5. It verifies Properties 1, 2, 3 and 4. Moreover, two nodes are not allowed to be connected by two edges with the same label.

Definition 11. (Node types) In order to describe the relationship between an eAG and a SeAG, we equip both graphs' nodes with two more values: a type value, and an identifier (see Definition 14). Within a SeAG, for two nodes v, v', we enforce that $type(v) = type(v') \Leftrightarrow v = v'$. Additionally, a type is associated to exactly one chronometer $c \in \mathscr{C}$.

Definition 12. (Simulation) Consider two rooted, directed labelled graphs $A = (V_A, E_A)$ and $B = (V_B, E_B)$. A simulation of B by A is a relation $R \in V_B \times V_A$ so that:
1. $(root(B), root(A)) \in R$ and
2. $(v_B, v_A) \in R \Rightarrow \forall v_B \lfloor e_B \rfloor v'_B \subseteq B, \exists (e_A, v_A) \in E_A \times V_A$ so that $v_A \lfloor e_A \rfloor v'_A \subseteq A \ \wedge \ label(e_A) = label(e_B) \ \wedge \ (v'_B, v'_A) \in R$. (Node-typed, rooted simulation) A node-typed simulation R verifies the above, plus $\forall (v_B, v_A) \in R, type(v_A) = type(v_B)$.

Example 4. Be A and B two rooted, connected graphs with one leaf. A simulates B implies that for all root-to-leaf path in B, there is a rooted path in A so that the sequences of labels along the two paths are equal. Conversely, given a graph A, building a graph B simulated by A restricts the possible label sequences along paths in the graph B:
Consider the graph A below, where the values in the nodes are their types. It is the Ott automaton [16] representing the regular expression $r =$`X:In(a|b)*X:Out`. The graph B, whose label sequences along the root to node paths are words from the language of r, is simulated by A.



The simulation is easy to decipher here : it is made out of the couples (v_B, v_A) so that $type(v_B) = type(v_A)$.

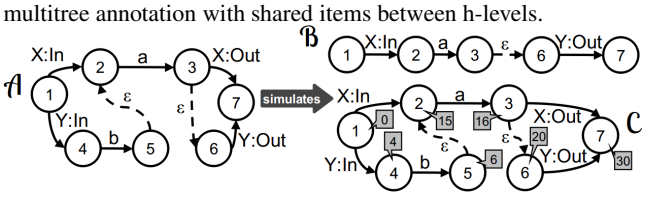
Definition 13. Be S an SeAG and G an eAG. S validates G iff there is a rooted, node-typed simulation of G by S and $type(leaf(G)) = type(leaf(S))$. In this case, G is called an *instance* of S.

Illustration, part 4. The Figure 3 shows a SeAG. Read as an automaton, it says that:
1. Within a Doc element are one or more Page elements. The backwards ε edge that connects the node typed 12 to the node typed 2 ensures multiplicity.
2. Within a Page are one or more Modules.
3. Modules are defined alternatively as containing either one Line or more (path through nodes typed 3-4-5-11), or containing one Figure (path through nodes typed 3-6-7-8-9-10-11). Alternatives are represented by parallel paths.
4. Within a Figure are either one or two Images. Optionality is represented by the alternative between an element or an ε edge (i.e. by parallel paths, in harmony with point 3. above).

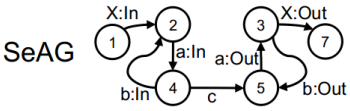
Discussion 1. (SeAG expressive power) As illustrated before, based on Ott's linear representation of regular expressions [16], a wide range of composite element contents can be expressed in an eAG: the | operator is represented by two parallel subpaths in the SeAG and the Kleene star operator by a backwards oriented ε edge; any combination is possible.

Importantly, the above automaton interpretation of SeAG only holds because of the well-formedness constraints for eAGs. For instance, the interpretation of the *cyclic* SeAG A in Example 4 as an Ott automaton, i.e. as a means to define an infinite set of *acyclic paths*, would not be correct without Property 5. Indeed, there is at least one cyclic graph that A simulates: A itself, but there is no way a graph structurally identical to A shall be an eAG. Consider A equipped with reference values on its nodes. Property 5, states that edges not suffixed `:LinkTo` go from nodes with a lower reference value to nodes with a higher one. Since the cyclic subgraph made out of the edges between the nodes typed 2 and 3 contains no `:LinkTo`, then whatever the reference values of its nodes, it cannot be part of an eAG. Conversely, the cycle in A will only be instantiated by acyclic paths (cf. graph B, Example 4) in the eAGs validated by A.

More generally, eAG data model and simulation-based validation make sense *together*. The following examples illustrate how the definition rules for eAG give sense to the SeAG formalism. First, an SeAG can express that two h-levels share elements: see graph A below. This SeAG does simulate the faulty graph B, where the inclusion semantics is lost (e.g. `X:Out` is missing), but since *for that reason*, regardless of its nodes references, B is not well-formed (see Property 2), it is not to be considered for validation. However, A *validates* the *well-formed* eAG C, which implements properly multitree annotation with shared items between h-levels.



Thanks to the same Property 2 in the eAG data model, an SeAG can contain recursive elements as well :



Discussion 2. (Caveats) [5, 1] point out several limitations to simulation-based validation. First, for a given instance graph, several schemas are eligible, since simulation is transitive. This matters greatly when schemas are inferred from the data, but does not when they are predefined.

Second, and more importantly, simulation-based validation as defined by [5] does not enforce the presence of a label. This is true for simulation between two general graphs. Still, this caveat can be bypassed by specifying an appropriate data model. Consider the SeAG A and the graphs B_1 and B_2 below. Even though A simulates both B_1 and B_2, it validates none: B_1 is not well-formed, and $type(leaf(B_2))$ is not equal to $type(leaf(A))$, which contradicts the validation definition.

Hence well-formedness and validation rules somehow enforce the presence of labels that simulation does not.

Last, simulation cannot prevent a node from having several outgoing edges. Said differently, as illustrated in Example 4, even when an SeAG contains one single (cyclic) path, there is no way to prevent the annotator to annotate the same content with several layers all instantiating the same path. Of course, this feature has positive aspects (e.g. self overlap is natively supported). But it means that a hierarchical SeAG will validate multitrees, not trees only.

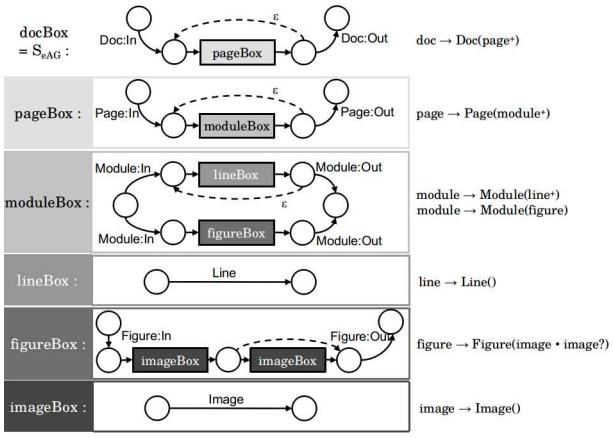


Figure 4: For all $n \in N$ as defined in Figure 5, each nBox representation (middle) and $R_n \subset R$ (right).

Still, there is a connexion between simulation and grammar-based validations. An XML document is, syntactically speaking, a tree; a (RelaxNG) schema is a Tree automaton [15], which validates the tree for which there is an "interpretation", as defined in Figure 5. We know there is a way to translate trees into eAGs. For instance, the eAG representing the tree X from Figure 5 is the arrowed path in Figure 2. There is also a way to derive a SeAG from a Tree automaton. Consider an automaton $TA = (S,N,T,R)$, for instance the one from Figure 5. In TA, T is the set of terminals. N is the set of non-terminals, among which are start symbols (S). The elements of R are called production rules. A rule associates a non terminal to a terminal, representing a possible labelled node of a tree, and a regular expressions over N against which the sons of the node shall match.

For our derivation of an SeAG from a TA, we enforce that if there is a rule $r = x \to X(reg) \in R$ so that reg can be expressed as $reg_1|reg_2$, with no common prefix and suffix between the words in the languages of reg_1 and reg_2, then r must be split into two rules $r_1 = x \to X(reg_1)$ and $r_2 = x \to X(reg_2)$. For instance, $module \to Module(line^+|figure)$ shall be split into two rules $module \to Module(line^+)$ and $module \to Module(figure)$. Then, the derivation of a SeAG S_{TA} from TA[10] defines as follows:

There is a partition of R into sets of rules sharing the same left-hand side. For any $n \in N$, let us call R_n one such subset of R. Then, every $n \in N$ may define what we call a unique Box, denoted nBox. The nBox is a rooted graph that reflects the content of the set of rules in R_n. In the nBox, each $r_i = n \to T_i(re_i) \in R_n$ is represented by a root-to-node path. If $re_i = \varnothing$, then the path is a single edge labelled T_i. Else, since re_i is a regular expression over N, it can be represented by the Ott automaton made out of the Boxes corresponding to re_i, escorted by two edges labelled T_i`:In` and T_i`:Out`. Figure 4 shows the nBoxes for the automaton in Figure 5. By replacing iteratively, in a bottom-up approach, the Boxes contained one in the others, we get a labelled graph which is S_{TA}. One can check that, in the case of Figure 4, this yields the SeAG shown on Figure 3.

[10] The following sketch leaves recursive element definition out.

Here comes the interesting point: this example illustrates that, given a tree automaton TA and a tree X so that there is an interpretation of X against TA, the SeAG derived from TA simulates the eAG representation of X[11].

The above provides a sketch of proof for the following connexion between interpretation and simulation for validation:

Property 6. Be a tree X and a tree automaton TA. Be G_X the eAG representation of X; be S_{TA} the SeAG derived from TA. Then, if there is an interpretation of X by TA, then S_{TA} simulates G_X.

4.3 Representation

So far we have introduced extended Annotation Graphs, a cyclic graph data model for multiple annotation of composite resources, along with a schema model, SeAG. Here, we consider the case where a schema is needed to proceed to annotation. We define a matrix representation for SeAG and eAG so that, given the representation of a schema, only valid instances can be represented. This is "validation by construction".

Definition 14. (Identifier sets) Be a graph $G=(V,E)$. There are two countable ordered sets $\mathscr{I}_G$ and $\mathscr{J}_G$ and two bijective functions $id:V\longrightarrow\mathscr{I}_G$ and $id:E\longrightarrow\mathscr{J}_G$ identifying the nodes and edges of G. The i^{th} element of the set $\mathscr{I}_G$, for instance, is denoted $[\mathscr{I}_G]_i$.

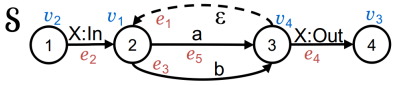
Definition 15. Be a graph $G=(V,E)$, $\mathscr{I}_G$ and $\mathscr{J}_G$ two sets of identifiers. Provided G contains no connected subgraph limited to a node and no loop, G can be represented by its incidence matrix $[G]^{\mathscr{I}_G,\mathscr{J}_G}$ so that, $\forall(i,j)\in\mathscr{I}_G\times\mathscr{J}_G$:

$$\begin{aligned}[G]^{\mathscr{I}_G,\mathscr{J}_G}_{i,j} &= 1 \text{ iff } \exists v\lfloor e\rfloor v'\subseteq G; id(v)=i \wedge id(e)=j\\ &= -1 \text{ iff } \exists v\lfloor e\rfloor v'\subseteq G; id(v')=i \wedge id(e)=j\\ &= 0 \text{ else.}\end{aligned}$$

Property 7. Be a SeAG $S=(V_S,E_S)$. Ordering the sets $\{type(v); v\in V_S\}$, $\{(type(v),label(e),type(v')); v\lfloor e\rfloor v'\subseteq S\}$, provides two special node and edge identifier sets $\mathscr{T},\mathscr{X}$.

PROOF. In a SeAG, no two nodes have the same type (Def. 11) or are connected by two edges with the same label (Def. 10). Any ordering of the sets is fine. □

Discussion 3. Consider the following SeAG :



The values v_i are possible identifiers for the nearby nodes, and e_j for edges, so that $\mathscr{I}_S=[v_1,v_2,v_3,v_4]$, for instance. Then, Property 7 means that it is possible to represent the incidence matrix of an SeAG S by indexing lines and columns either on any $\mathscr{I}_S\times\mathscr{J}_S$ or on $\mathscr{T}\times\mathscr{X}$ in particular. For instance, when indexed over $\mathscr{T}\times\mathscr{X}$:

$$[S]^{\mathscr{T},\mathscr{X}}=\begin{array}{c} \\ 1\\2\\3\\4\end{array}\begin{array}{c}\begin{array}{ccccc}{}_1\texttt{X:In}_2 & {}_2a_3 & {}_2b_3 & {}_3\varepsilon_2 & {}_3\texttt{X:Out}_4\end{array}\\\begin{bmatrix}1&0&0&0&0\\-1&1&1&-1&0\\0&-1&-1&1&1\\0&0&0&0&-1\end{bmatrix}\end{array}$$

It is also possible to express a *subgraph* of S in an incidence matrix indexed over the full identifier sets. For instance, below is the incidence matrix over $\mathscr{I}_S$ and $\mathscr{J}_S$ of $H=\{v\lfloor e\rfloor v'\subseteq S; (type(v),label(e),type(v'))=(2,b,3)\}$, subgraph of S :

$$[H]^{\mathscr{I}_S,\mathscr{J}_S}=\begin{array}{c} \\ v_1\\v_2\\v_3\\v_4\end{array}\begin{array}{c}\begin{array}{ccccc}e_1 & e_2 & e_3 & e_4 & e_5\end{array}\\\begin{bmatrix}0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&-1&0&0\end{bmatrix}\end{array}$$

[11] We obliterate the question of node types here. We only compare the bare simulation and interpretation relations.

Definition 16. Given a $n\times m$ matrix $[M]$ of integers, the positive restriction of $[M]$ is the $n\times m$ matrix $[M^+]$ so that $\forall i,j$, $[M^+]_{i,j}=[M]_{i,j}$ iff $[M]_{i,j}>0$, else $[M^+]_{i,j}=0$. The definition of $[M^-]$ the negative restriction of $[M]$ is natural.

Discussion 4. Consider two graphs G and H, $H\subseteq G$ and the incidence matrix $[H]^{\mathscr{I}_G,\mathscr{J}_G}$. Then the positive restriction of $[H]^{\mathscr{I}_G,\mathscr{J}_G}$, read column by column, lists the identifiers of the nodes that are the summits of the edges of H whose identifier matches the one of the column. Conversely, the negative restriction of $[H]^{\mathscr{I}_S,\mathscr{J}_S}$ defined in Discussion 3 is :

$$[H^-]^{\mathscr{I}_S,\mathscr{J}_S}=\begin{array}{c} \\ v_1\\v_2\\v_3\\v_4\end{array}\begin{array}{c}\begin{array}{ccccc}e_1 & e_2 & e_3 & e_4 & e_5\end{array}\\\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&-1&0&0\end{bmatrix}\end{array}$$

Note that the sum of the positive and negative restriction of any incidence matrix gives the incidence matrix.

Definition 17. (Template) Be S a SeAG, G a graph that can be represented by its incidence matrix, and $\mathscr{I}_G$, $\mathscr{J}_G$ identifier sets for G. Consider the block-matrix obtained by replacing each value $s_{i,j}$ of $[S]^{\mathscr{T},\mathscr{X}}$ by a matrix $[M_{i,j}]$, so that:
- $s_{i,j}=0\Rightarrow[M_{i,j}]=[\varnothing]^{\mathscr{I}_G,\mathscr{J}_G}$, where $\varnothing$ is the empty graph, whose incidence matrix is always zero ;
- $s_{i,j}=1\Rightarrow[M_{i,j}]=[A]$, where $[A]$ is the positive restriction of the incidence matrix over $\mathscr{I}_G$, $\mathscr{J}_G$ of $H_j\subseteq G$, with $H_j=\{v\lfloor e\rfloor v'\subseteq G; (type(v),label(e),type(v'))=[\mathscr{X}]_j\}$;
- $s_{i,j}=-1\Rightarrow[M_{i,j}]=[B]$, where $[B]$ is the negative restriction of the incidence matrix over $\mathscr{I}_G$, $\mathscr{J}_G$ of H_j.

This block-matrix is called the expression of G on the template of S, denoted $[G/Temp.S]$.

Example 5. Consider the SeAG S defined in Discussion 3. The expression of S on its own template is:

$$[S/Temp.S]=\begin{array}{c} \\ 1\\2\\3\\4\end{array}\begin{array}{c}\begin{array}{ccccc}{}_1\texttt{X:In}_2 & {}_2a_3 & {}_2b_3 & {}_3\varepsilon_2 & {}_3\texttt{X:Out}_4\end{array}\\\begin{bmatrix}[A_1]&0&0&0&0\\[B_1]&[A_2]&[A_3]&[B_4]&0\\0&[B_2]&[B_3]&[A_4]&[A_5]\\0&0&0&0&[B_5]\end{bmatrix}\end{array}$$

with, for instance, $[A_3]=[H^+]^{\mathscr{I}_S,\mathscr{J}_S}$ and $[B_3]=[H^-]^{\mathscr{I}_S,\mathscr{J}_S}$ as defined in Discussion 4.

Definition 18. Be S an SeAG and $G,\mathscr{I}_G,\mathscr{J}_G$ a graph containing no subgraph limited to a node and no loop, along with two sets of identifiers. G is said to be fully expressible on the template of S, denoted $G\triangleleft[Temp.S]$, iff the sum of the inner matrices of $[G/Temp.S]$ is equal to $[G]^{\mathscr{I}_G,\mathscr{J}_G}$ the incidence matrix of G, indexed over the same sets as the inner matrices of $[G/Temp.S]$.

Property 8. Be S an SeAG. Then S is fully expressible on its own template.

PROOF. $\forall l\in[0;|\mathscr{X}|[$, the l^{th} column of $[S/Temp.S]$ contains two matrices $[A_l]$ and $[B_l]$. Since they are respectively the positive and negative restrictions of the incidence matrix of $H_l\subseteq S$, which is the union of all the subgraphs $v\lfloor e\rfloor v'$ characterized by the same triple $[\mathscr{X}]_l$ of types and label, $[A_l]+[B_l]=[H_l]^{\mathscr{I}_S,\mathscr{J}_S}$. Since $\mathscr{X}$ is the set of possible triples for S, $\sum_{0\leq l<|\mathscr{X}|}[H_l]^{\mathscr{I}_S,\mathscr{J}_S}=[S]^{\mathscr{I}_S,\mathscr{J}_S}$. □

Importantly, only schemas define a template. In particular, given an instance G and any identifier sets $\mathscr{I}$, $\mathscr{J}$, since there may not be bijections between those sets and $\mathscr{T},\mathscr{X}$, the notion of template of G is undefined. Still, it is possible to try to express G, not over its own template, *but over the template of a given schema S*.

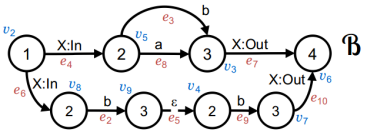
Let us denote this representation $[G/Temp.S]$. The schema defines the template, that is, the outer matrix of $[G/Temp.S]$, indexed over $\mathscr{T} \times \mathscr{X}$: it restricts the types, the labels between two given types and the paths along which those labels may occur. Be then $[\mathscr{X}]_l \in \mathscr{X}$. Just like above, we can define $H_l = \{v\lfloor e\rfloor v' \subseteq G; (type(v), label(e), type(v')) = [\mathscr{X}]_l\}$, so that $H_l \subseteq G$. Then the inner matrices of $[G/Temp.S]$ are defined just the same way as those in $[S/Temp.S]$, that is: in the l^{th} outer column, on the right outer lines, as the positive and negative restrictions of $[H_l]^{\mathscr{I},\mathscr{I}}$ (see Definition 17).

Interestingly, this approach can be taken for any graph G and any schema S. If the graph contains no edge conforming the schema, then $[G/Temp.S]$ is null. On the contrary, an important result is that provided G is an instance of S, then G is *fully expressible* on $[Temp.S]$. We can even go further:

Property 9. Be a SeAG S and an eAG G. Then S validates G iff $G \triangleleft [Temp.S]$ and $type(leaf(G)) = type(leaf(S))$ and $type(root(G)) = type(root(S))$.

PROOF. $\Rightarrow$: S validates G, then the types of the two graphs' leafs are equal, by definition of validation. Idem for the roots. The fact that validation implies $G \triangleleft [Temp.S]$ can be proven just like Property 8, with one more argument. The fact that the sum of the two inner matrices characterised by the same $L \in \mathscr{X}$ yields the incidence matrix of the union of all the subgraphs $v\lfloor e\rfloor v' \subseteq G$ characterised by L holds. Yet, it has to be proven that there is no subgraph $v\lfloor e\rfloor v' \subseteq G$ so that $(type(v), label(e), type(v')) \notin \mathscr{X}$. Since G is rooted and connected, one can check that the presence of such a subgraph shall contradict the existence of a rooted simulation.
$\Leftarrow$: Be $G = (V,E)$, $S = (V_S, E_S)$. $G \triangleleft [Temp.S]$ implies that $\forall v\lfloor e\rfloor v' \subseteq G$, $\exists L \in \mathscr{X}$ so that $(type(v), label(e), type(v')) = L$, which means $\exists! v_S\lfloor e_S\rfloor v'_S \subseteq S$ so that $type(v_S) = type(v)$, $type(v'_S) = type(v')$ and $label(e_S) = label(e)$.
This defines two functions $\delta : V \to V_S$ and $\delta_E : E \to E_S$ so that $\forall v\lfloor e\rfloor v' \subseteq G$, $\exists!(\delta(v), \delta_E(e), \delta(v')) \in V_S \times E_S \times V_S$ so that $\forall x, type(\delta(x)) = type(x)$, $\forall y, label(\delta_E(y)) = label(y)$ and $\delta(v)\lfloor\delta_E(e)\rfloor\delta(v') \subseteq S$.
Additionally, the fact that $type(root(G)) = type(root(S))$ implies $\delta(root(G)) = root(S)$. Then $D = \{(v, \delta(v)); v \in V\}$ is a rooted, node-typed simulation of G by S. □

Illustration, part 5. Consider the eAG B from Example 4. Let us equip its nodes and edges with identifiers, as shown below.



The representation of B in $[Temp.S]$ is:

$$[B/Temp.S] = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} {}_1\texttt{X:In}_2 & {}_2a_3 & {}_2b_3 & {}_3\varepsilon_2 & {}_3\texttt{X:Out}_4 \end{array} \\ \begin{bmatrix} [A_1] & 0 & 0 & 0 & 0 \\ [B_1] & [A_2] & [A_3] & [B_4] & 0 \\ 0 & [B_2] & [B_3] & [A_4] & [A_5] \\ 0 & 0 & 0 & 0 & [B_5] \end{bmatrix} \end{array}$$

with $[B_3]$ the negative restriction of $[H]^{\mathscr{I}_B,\mathscr{I}_B}$, for instance, for $H = \{v\lfloor e\rfloor v' \subseteq B; (type(v), label(e), type(v')) = (2,b,3)\}$:

$$[B_3] = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{array} \begin{array}{c} \begin{array}{cccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Now compare $[B/Temp.S]$ with $[S/Temp.S]$ as detailed in Example 5. The two matrices share the same outer matrix, which is descriptive of S, they only differ by the values of the inner matrices (e.g. see the value of $[B_3]$ for S in Example 5). Based on Property 9, we can finally conclude:

Given a schema S, the eAGs it validates are the well-formed eAGs model that can be fully expressed in $[Temp.S]$, and whose root and leaf types respect those of S. This means that an instance of S is an eAG *that can be described by* the set of matrix values that fill $[Temp.S]$. From a manufacturing point of view, if the annotator of a resource is given means (through an ergonomic HCI) to define the matrix values corresponding to $[Temp.S]$, in a way that ensures well-formedness, then, by construction, the resulting graph will be valid against the schema. This meets the goal of providing on-the-fly validation for M-S data.

5. CONCLUSION

In this paper, we introduce eAG, an extension of Annotation graphs, along with a novel schema model based upon the notion of simulation. A dedicated representation for eAGs and schemas enables to proceed to validation "by construction": provided a schema, only valid eAGs can be expressed, which bypasses the algorithmic cost of traditional approaches for validation of graph-structured data.

Still, the eAG data model is not restricted to this use case, and simulation-based validation can be adapted to the situations where any eAG $G = (V,E)$ is confronted to any SeAG $S = (V_S, E_S)$. First case, G was made according to a schema $S' = (V_{S'}, E_{S'})$, and the question is whether it conforms to S or not. By transitivity of simulation, S validates G iff S simulates S' so that $(leaf(S'), leaf(S))$ are in the simulation (indicating, *modulo* retyping the nodes of S, a node-typed simulation of S' by S). This checks in $O(|V_{S'} \cup V_S| \cdot |E_{S'} \cup E_S|)$ [19]. Second case, G was not made according to any schema. In this case, node types are irrelevant. An adaptation of SeAG validation is: S validates G iff there is a (general) simulation $D \subseteq V \times V_S$ so that $\forall v \in V$, $\exists! v_S \in V_S$ so that $(v, v_S) \in D$ (the uniqueness of v_S for each v defines a typing of the nodes of G according to S). This checks in $O(|V \cup V_S| \cdot |E \cup E_S|)$. In both cases, this is a reasonable cost for a cyclic graph-based data model.

6. ACKNOWLEDGMENTS

This work is supported by the ARC5 program of the Rhône-Alpes region, France.

7. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[2] S. Bird and M. Liberman. A formal framework for linguistic annotation. *Speech communication*, 33(1):23–60, 2001.

[3] E. Bruno and E. Murisasco. Describing and querying hierarchical xml structures defined over the same textual data. In *Proceedings of the 2006 ACM symposium on Document engineering*, pages 147–154. ACM, 2006.

[4] E. Bruno and E. Murisasco. An xml environment for multistructured textual documents. In *Digital Information Management, 2007. ICDIM'07. 2nd International Conference on*, volume 1, pages 230–235. IEEE, 2007.

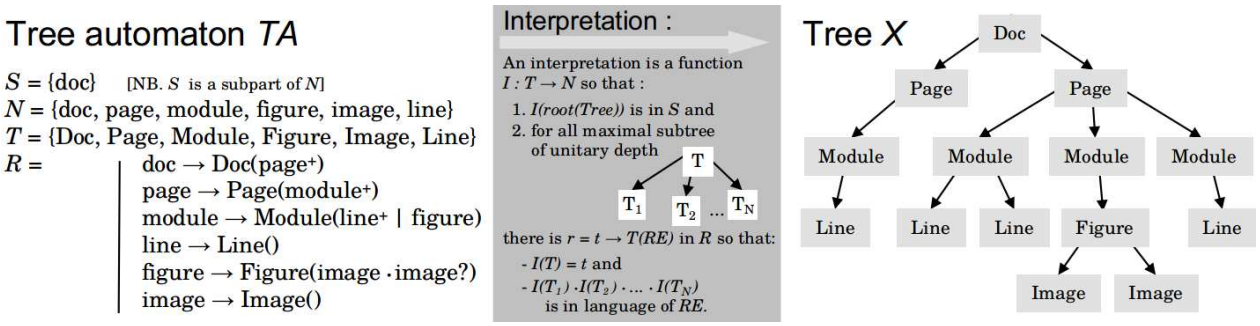


Figure 5: RelaxNG tree automaton-based XML validation mechanism.

[5] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Database Theory -ICDT'97*, pages 336–350. Springer, 1997.

[6] L. Burnard and S. Bauman. *TEI P5: Guidelines for electronic text encoding and interchange*. TEI Consortium, 2008.

[7] H. A. Cayless. Rebooting tei pointers. *Journal of the Text Encoding Initiative*, (6), 2013.

[8] A. Di Iorio, S. Peroni, and F. Vitali. Using semantic web technologies for analysis and validation of structural markup. *International Journal of Web Engineering and Technology*, 6(4):375–398, 2011.

[9] C. F. Goldfarb and Y. Rubinsky. *The SGML handbook*. Oxford University Press, 1990.

[10] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. 1997.

[11] M. Hilbert, A. Witt, and O. Schonefeld. Making concur work. In *In Extreme Markup Languages*, 2005.

[12] C. Huitfeldt and C. Sperberg-McQueen. Texmecs: An experimental markup meta-language for complex documents. *URL http://www. hit. uib. no/claus/mlcd/papers/texmecs. html*, 2001.

[13] H. Jagadish, L. V. Lakshmanan, M. Scannapieco, D. Srivastava, and N. Wiwatwattana. Colorful xml: one hierarchy isn't enough. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 251–262. ACM, 2004.

[14] D. Lee, M. Mani, and M. Murata. Reasoning about xml schema languages using formal language theory. Technical report, Citeseer, 2000.

[15] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of xml schema languages using formal language theory. *ACM Transactions on Internet Technology (TOIT)*, 5(4):660–704, 2005.

[16] G. Ott and N. H. Feinstein. Design of sequential machines from their regular expressions. *Journal of the ACM (JACM)*, 8(4):585–600, 1961.

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

[18] P.-É. Portier, N. Chatti, S. Calabretto, E. Egyed-Zsigmond, and J.-M. Pinon. Modeling, encoding and querying multi-structured documents. *Information Processing & Management*, 48(5):931–955, 2012.

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

[20] D. Reynolds, C. Thompson, J. Mukerji, and D. Coleman. An assessment of rdf/owl modelling. *Digital Media Systems Laboratory, HP Laboratories Bristol*, 28, 2005.

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

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

[23] M. Stührenberg and C. Wurm. Refining the taxonomy of xml schema languages. a new approach for categorizing xml schema languages in terms of processing complexity. In *Proceedings of Balisage: The Markup Conference*, volume 5, 2010.

[24] D. Suciu. Semistructured data and xml. In *Information organization and databases*, pages 9–30. Springer, 2000.

[25] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in owl. In *AAAI*, 2010.

[26] J. Tennison. Creole: Validating overlapping markup. In *Proceedings of XTech*, 2007.

[27] J. Tennison and W. Piez. The layered markup and annotation language (lmnl). In *Extreme Markup Languages*, 2002.