Rappresentare overlap in XML

1 Introduzione

Overlap é il termine comunemente utilizzato per descrivere i casi in cui piú strutture di markup non si annidano in maniera ordinata: in numerosi domini, durante il processo di mark up di un documento sono frequenti i casi in cui lo stesso frammento di contenuto debba essere associato a markup descriptor diversi e talvolta incompatibili. Il caso piú semplice é quello in cui una porzione di un elemento si sovrappone (da cui il nome overlap) con un'altra, ad esempio la parola "likes" nel frammento seguente:

```
<doc><b>John <i>likes</b> Mary</i></doc>
```

Sono numerosi i domini in cui si verificano situazioni di questo tipo, ad esempio nell'analisi letteraria, nella linguistica computazionale, durante la revisione e il commento di testi da parte di piú autori, in contesti in cui si vuole tenere traccia dei cambiamenti dei documenti, ecc.

Osservando gli esempi precedenti risulta evidente come il problema dell'overlapping markup sia molto comune, ed emerge il fatto che debba essere visto come una caratteristica intrinseca alle strutture dei documenti di testo in specifici domini. Una parte consistente della letteratura sul markup é dedicata al problema dell'overlap, e sono numerosi gli eventi scientifici specializzati sull'argomento. In seguito a questo lavori, negli ultimi anni sono emerse proposte di approcci estremamente vari per gestire queste situazioni.

Alcuni di questi approcci sono proposti a livello di linguaggio: numerosi linguaggi della famiglia XML aggiungono specifiche strutture di markup language-dependent per il supporto dell'overlap. Il problema principale da affrontare in questo contesto é che il data model di SGML e XML impone a livello di grammatica una gerarchia di contenimento che genera un singolo albero, in cui non é consentita alcuna situazione di overlap (il codice precedente ad esempio rappresenta un frammento XML non ben formato).

Per risolvre questo problema vengono usati elementi o tag specifici per esprimere la semantica dell'overlap in maniera implicita. L'esempio piú rappresentativo é TEI, che nella specifica di un linguaggio di markup (in XML) per la rappresentazione dei testi in formato digitale suggerisce una serie di workarounds (trucchi) per gestire l'overlap, specificando l'uso di costrutti SGML/XML per forzare gerarchie multiple all'interno di una singola gerarchia tree-based.

L'obiettivo di queste pagine é di presentare le principali tecniche per la gestione dell'overlap. Per fare questo vengono dapprima classificate le caratteristiche dei testi non esprimibili in una singola struttura: queste sono le principali feature che sono utilizzate per confrontare le tecniche presentate in seguito, e per misurare quanto queste siano efficaci nella gestione di situazioni di overlap.

Successivamente vengono descritte le principali tecniche per la gestione dell'overlap, mettendo in luce i principali vantaggi e svantaggi di ogni soluzione. Infine é presente una tabella riassuntiva, per agevolare il confronto delle varie soluzioni. Nel corso di questa trattazione é riposta un'attenzione particolare alle soluzioni specifiche presentate da **TEI**, in quanto esempio rappresentativo di queste tecniche.

2 Caratteristiche di documenti di testo complessi

L´insieme delle caratteristiche di documenti non esprimibili in una singola struttura ad albero puó essere riassunto come segue:

1. **classic overlap:** é il caso in cui due frammenti di documento, che devono venire annotati con general identifiers diversi (i nomi dei tag di XML), si sovrappongono gli uni agli altri. Tipici esempi sono gli scenari in cui é necessario identificare gerarchie multiple concorrenti sullo stesso documento: é il caso in cui si intende codificare lo stesso documento in maniera

tale da catturare aspetti multipli e spesso indipendenti, come strutture sintattiche (ad esempio il riconoscimento delle parti del discorso, della struttura grammaticale, fonetica), dichiarazioni tipografiche, analisi semantiche o analisi relazionali (ad esempio connessioni con dati e modelli esterni). Quando si tenta di unire tali gerarchie in un singolo documento é probabile che una struttura si sovrapponga ad un´altra. Per esempio un paragrafo puó iniziare in una pagina e terminare all´interno della successiva.

2. **self overlap:** ci sono anche casi in cui a sovrapporsi sono due elementi della stessa struttura e che hanno lo stesso nome. Classici esempi di questo tipo sono i link ipertestuali o annotazioni che si sovrappongono. Consideriamo ad esempio il secondo caso: questa situazione potrebbe verificarsi in un contesto in cui piú revisori debbano commentare lo stesso testo. In questi casi é ragionevole considerare questi commenti in overlap come appartenenti alla stessa struttura (la struttura dei commenti). Una gestione banale di questi casi rende impossibile distinguere fra situazioni di overlap da annidamenti corretti degli elementi, come ad esempio

   `<comment>`John `<comment>`likes `</comment>`Mary`</comment>`

3. **virtual elements:** é un elemento che non é presente in maniera esplicita all´interno del testo, ma la cui presenza puó essere inferita da un´applicazione in funzione di una codifica fornita. Questi elementi vengono utilizzati per definire elementi il cui contenuto é il risultato della riorganizzazione di materiale presente altrove all´interno del documento.
   TEI ne specifica due tipi:
   i. virtual element ottenuti semplicamente *clonando* elementi esistenti. Esistono due codifiche diversi che rientrano in questo caso:
      - identical elements: usando l´attributo @sameAs é possibile documentare il fatto che due elementi hanno il medesimo contenuto.
      - Virtual copies: con l´attributo @copyOf si puó indicare in maniera simile che il contenuto di un elemento é identico a quello di un altro. La differenza risiede nal fatto che il contenuto in questo caso non viene ripetuto: eventualmente sará compito di un´applicazione sostituire il contenuto di un elemento (che presenta l´attributo @copyOf) con il contenuto dell´elemento specificato nell´attributo.

   ii. Virtual element costituiti dall´*aggregazione* di elementi esistenti (vedi par. 3.3): a causa di una organizzazione gerarchica restrittiva degli elementi, o per ragioni differenti, puó non essere sempre possibile o desiderabile includere tutte le parti eventualmente frammentate di un segmento di testo all´interno di un singolo elemento. Le soluzioni possibili sono:
      - Intermediate pointers (vedi sez. 16.1.4 TEI P5)
      - `<join>` element: identificano un segmento di testo frammentato, puntando agli elemento (eventualmente discontinui) che lo compongono. É possibile esprimere questo tipo di elementi in maniera equivalente utilizzando un elemento `<link>` di tipo ¨join¨ (attributo `@type`).
      - Linking attribute per l´aggregazione sono `@next` e `@prev` (puntano rispettivamente all´elemento successivo/precedente di un aggregato virtuale di cui l´elemento in questione é parte).

4. **containment/dominance decoupling:** é utile distinguere fra strutture che contengono altre strutture e strutture che dominano altre strutture. Il contenimento é una relazione casuale fra range, mentre la dominanza é una relazione che ha una semantica precisa. É possibile che una pagina contenga una strofa, mentre una poesia domina sempre le strofe che contiene.
   Dominance: é la chiusura transitiva della relazione parent/child
   É una relazione fra le porzioni di un documento in cui una parte si dice dominare un´altra se é un suo antenato nella struttura ad albero del documento.
   Containment: é la relazione superset/subset sui nodi foglie raggiungibili da un nodo seguendo gli archi nel verso parent/child.

Il contenimento si ottiene osservando due parti di documento dal punto di vista di quali fette/porzioni dei caratteri contenuti nel documento essi racchiudano: una parte di documento contiene un´altra se contiene/racchiude tutti i caratteri dell´altra parte.
I linguaggi di markup basati su una struttura ad albero come XML tendono ad avere la proprietá che il contenimento implica la dominanza, mentre spesso questo aspetto non viene richiesto. Pensando a gerarchie multiple che condividono lo stesso contenuto di caratteri, non é per niente evidente il motivo per cui parti appartenenti a una gerarchia debbano essere messe in relazione via dominanza con parti strutturali appartenenti ad altre gerarchie.

Gerarchie sacre e profane

Un altro aspetto da tenere in considerazione durante l´analisi di documenti costituiti di gerarchie multiple, é la distinzione dei nodi costituenti tali gerarchie in sacri e nodi.

I nodi sacri sono condivisi da tutte le gerarchie in overlap, mentre i nodi profani sono specifici di una particolare gerarchia. Consideriamo il caso piú comune di overlap: l´uso di due o piú vocabolari indipendenti per strutturare lo stesso contenuto. In questo caso, il markup del contenuto condiviso é inteso come sacro, mentre il markup di ogni singolo vocabolario é profano. Non é permesso overlap all´interno del/fra markup sacro, né all´interno di una singola gerarchia profana, né nell´unione fra markup sacro e un singolo vocabolario profano. Al contrario, é possibile che si presenti overlap quando si uniscono due o piú vocabolari profani nello stesso documento.

3 Workarounds – rappresentazione dell´overlap in XML

Le sintassi XML, come ad esempio TEI o LMNL, utilizzano diverse tecniche per gestire overlapping conosciute con il termine workarounds (trucchi): si tratta di nascondere informazioni strutturali sotto la forma di qualcos'altro: divisione di elementi individuali, elementi vuoti che esprimono limiti, riferimenti indiretti, ecc. Tutte queste soluzioni codificano informazioni strutturali secondarie in modo da non rompere o offuscare la gerarchia principale che viene espressa nella struttura XML visibile. Questo consente di gestire gerarchie multiple, ma generalmente il prezzo da pagare é la maggiore difficoltá di trovare, identificare e operare sulle strutture aggiunte sopra la gerarchia XML principale. In seguito vengono presentate i cinque principali tipi di trucchi descritti in letteratura.

3.1 TEI Milestone

É un metodo per codificare strutture in overlap in XML: consiste nell´identificare un vocabolario come primario che viene rappresentato con una gerarchia standard XML, e nell´usare elementi vuoti per delimitare l´inizio e la fine dei tag dei vocabolari secondari. In questo modo é possibile gestire elementi che non risultano annidati in maniera corretta all´interno di un documento XML, impedendo di invalidare un documento, ma riuscendo comunque a identificare l´inizio e la fine dei tag per eleborazioni successive.

Questa tecnica viene descritta in dettaglio in TEI P5 guidelines: viene specificato uno speciale elemento vuoto **<milestone/>** che ha la funzione di marcare/segnare/notare la posizione all´interno di un testo in cui una categoria di un sistema di riferimento cambia. Questi elementi non hanno contenuto, ma semplicemente suddividono il testo in regioni, allo stesso modo in cui una pietra miliare (milestone) segna dei punti lungo una strada, dividendola in maniera implicita in segmenti. **TEI** fornisce una serie di elementi milestone che possono essere usati per segnare l´inizio di una particolare feature del testo:

<milestone/>	segna un estremo che separa sezioni di testo, tipicamente ma non necessariamente per indicare un punto in cui una parte di un sistema di riferimento standard si modifica, quando il cambiamento non é rappresentato da un elemento strutturale.
<pb/> (page break) **<lb/>** (line break) **<cb/>** (column break)	marcano il limite di pagine, linee e colonne.
<handshift/>	Indica l´inizio di una sequenza di testo scritta da un'altra persona (la mano della stesura di una parte del testo o di una particolare caratteristica del documento)

L´attributo globale @n viene usato con gli elementi <pb/>, <lb/> e <cb/> per fornire un valore per una particolare unitá associata ad una milestone (per esempio, il numero di pagina o di linea). Poiché non é strutturale, la validazione di un sistema di riferimento basato su milestone non puó venire facilmente verificato da un parser XML, per cui é responsabilitá di colui che codifica o di un´applicazione software assicurare che vengano forniti nell´ordine corretto.

Esempio: usando <lb/> é possibile indicare sia l'allineamento fisico di una poesia all'interno della pagina, sia la divisione grammaticale in frasi

```
<p>
    <seg>
        <lb n="1"/>Scorn not the sonnet;</seg>; <seg>critic, you have
         frowned, <lb n="2"/>Mindless of its just honours;</seg>
    <seg>with this
         key <lb n="3"/>Shakespeare unlocked his heart;</seg>
    <seg>the melody
    <lb    n="4"/>Of this small lute gave ease to Petrarch's
     wound.</seg>
</p>
```

L'uso di questi elementi é conforme alle specifiche TEI. Il significato di questi elementi é riservato: semanticamente <lb/> indica l'inizio di una nuova linea (a livello tipografico). Per questo motivo l'utilizzo di questi elementi in alcuni contesti risulta appropriato (ad esempio nella poesia moderna la divisione delle linee a livello tipografico e metrico corrispondono), mentre in altri casi non risulta adatto (nei manoscritti anglosassoni antichi, per esempio, la fine di una linea non viene usata per indicare la lineazione a livello metrico).

Per ovviare a questo problema, é possibile utilizzare l'elemento **<anchor/>**, che permette di marcare il limite di segmenti generici di testo. In questo caso é possibile indicare il tipo di feature da delimitare e se una particolare istanza apra o chiuda tale feature.

```
<l>
    <anchor subtype="sentenceStart" type="delimiter"/>
    Scorn not the sonnet;
    <anchor subtype="sentenceEnd" type="delimiter"/>
    <anchor subtype="sentenceStart" type="delimiter"/>
    critic, you have frowned,
</l>
<l> Mindless of its just honours;
    <anchor subtype="sentenceEnd" type="delimiter"/>
    <anchor subtype="sentenceStart" type="delimiter"/> with this key
```

```
</l>
<l> Shakespeare unlocked his heart;
    <anchor subtype="sentenceEnd" type="delimiter"/>
    <anchor subtype="sentenceStart" type="delimiter"/> the melody
</l>
<l> Of this small lute gave ease to Petrarch's wound.
    <anchor subtype="sentenceEnd" type="delimiter"/>
</l>
```

Questo metodo é conforme alla specifica TEI.

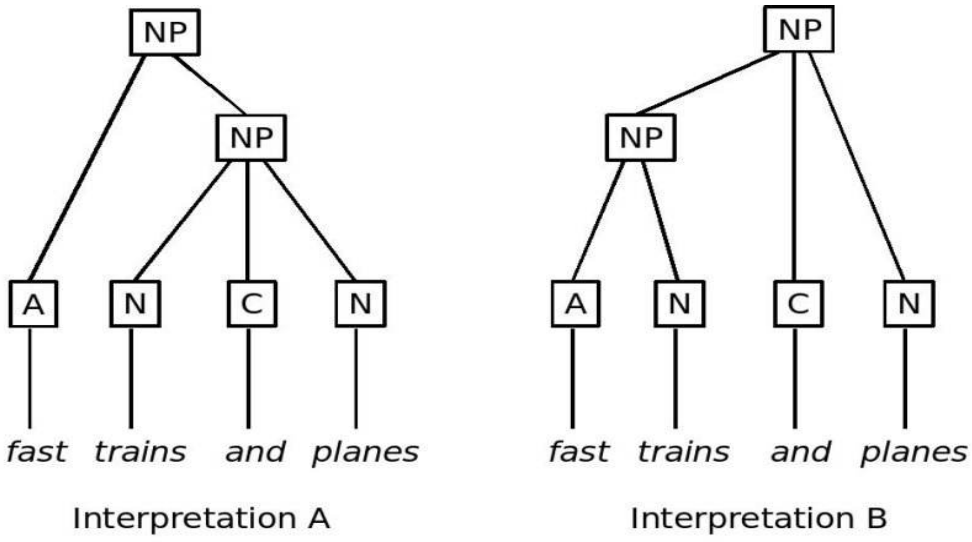
Un **altro approccio** consiste nel progettare elementi personalizzati che forniscano informazioni piú ricche sulle feature che delimitano o sui limiti stessi.. Queste informazioni possono essere incluse come valori di attributi o come parti del nome stesso dell'elemento , per esempio:

```
<boundaryStart element="sentence"/>...<boundaryEnd element="sentence"/>,
<sentenceBoundary position="start"/>...<sentenceBoundary position="end"/>,
<sentenceBoundaryStart/>...<sentenceBoundaryEnd/>
```

Se questi elementi possono essere sostituiti da elementi TEI e attributi senza perdita di informazioni, questo metodo é conforme a TEI, se invece introduce informazioni o distinzioni che non possono essere espresse usando elementi TEI standard, il metodo é una estensione.

Infine, gli elementi che generalmente vengono usati per codificare feature testuali annidate (per esempio `said`, `seg`, `l`) possono essere adattati a svolgere il ruolo di delimitatori vuoti di segmenti, in modo particolare quando le feature che codificano si estendono oltre i limiti gerarchici. A questi elementi vengono aggiunti attributi (ad esempio **sID** e **eID**) per consentire di collegare in maniera non ambigua i punti di inizio e di fine (**co-indexing**). Questo metodo é stato introdotto in letteratura con vari nomi, incluso Trojan milestone, HORSE markup, CLIX e COLT.
Attraverso questo meccanismo é possibile codificare anche situazioni di **self-overlap**, ma non consente di eprimere **virtual elements**. Senza questo accorgimento inoltre sarebbe impossibile distinguere fra elementi effettivamente vuoti ed elementi vuoti utilizzati come milestone.

In ognuno degli esempi precedenti tranne l'ultimo, la relazione fra delimitatore iniziale e finale (qualora esista) di una specifica feature é implicita: si assume che il delimitatore di fine chiuda il delimitatore di inizio precedente piú prossimo o, nel caso di milestone, che segni sia la fine dell'istanza precedente che la l'inizio di quella successiva. La questione diventa piú complessa ancora nel caso in cui porzioni di testo non annidate siano in overlap con dell'altro testo non annidato dello stesso tipo. Ad esempio, svolgendo un'analisi grammaticale delle possibili interpretazioni della frase fast trains and planes: in questo caso, é possibile intendere l'aggettivo fast come associato a trains and plans, oppure solamente a trains.



Esempio: presentiamo un metodo non ambiguo per codificare le diverse analisi della frase, che permette di associare liberamente i limiti iniziali e finali dei segmenti

```
<phr function="NP">
     <anchor type="delimiter" subtype="NPstart" xml:id="NPInterpretationB"/>
     <w function="A">Fast</w>
     <anchor type="delimiter" subtype="NPstart" xml:id="NPInterpretationA"/>
     <w function="N">trains</w>
     <anchor type="delimiter" subtype="NPend" corresp="#NPInterpretationB"/>
     <w function="C">and</w>
     <w function="N">planes</w>
     <anchor type="delimiter" subtype="NPend" corresp="#NPInterpretationA"/>
</phr>
```

In questa codifica, la prima interpretazione, in cui fast modifica un NP contenente trains and planes, NP viene aperto usanso un tag `<anchor>` con `@xml:id` il valore NPInterpretationA e chiuso con un tag `<anchor>` con lo stesso valore come valore dell'attributo `@corresp`; nella seconda interpretazione, in cui fast e trains sono contenuti in NP, NP viene aperto usando un tag `<anchor>` con `@xml:id` il valore NPInterpretationB e chiuso con un tag <anchor> con lo stesso valore come valore dell'attributo `@corresp`.

Nonostante i vantaggi sopramenzionati, i marcatori di limite hanno lo svantaggio di richiedere un'**elaborazione successiva**: poiché gli elementi in questione (per esempio frasi di poesie, o semplici frasi come l'esempio precedente) non sono rappresentate in maniera uniforme dai nodi nell'albero del documento, é necessario l'utilizzo di software specifico che riconosca gli elementi e ne ricostruisca la struttura. É da osservare come questo approccio sia comlicato e, di conseguenza, incline agli errori.

Una questione ancora piú importante é che questo metodo nasconde la relazione fra inizio e fine di un elemento logico. Ció rende impossibile ai software standard l'esecuzione dello stesso tipo di **validazione** possibile nelle codifiche standard. Usando linguaggi di schema basati su grammatiche (per esempio DTD, W3CSchema e RELAX NG) non é possibile definire un content model per range limitati da elementi vuoti. Linguaggi di schema basati su regole (per esempio Schematron) possono invece essere usati per definire ulteriori restrinzioni: ad esempio é possibile legittimare o proibire la presenza di sequenze di certi elementi fra elementi vuoti.

Un altro svantaggio di questa codifica sta nel fatto che, poiché tutte le gerarchie vengono fuse in un singolo documento XML, l'ordine lessicografico del documento e le regole di parsing XML possono creare relazioni di dominanza far nodi profani appartenenti a gerarchie diverse, anche se non fosse intenzione dell'autore esprimere questa relazione. L'unico modo per **disaccoppiare** le relazioni di **dominanza** da relazioni di **contenimento** é fornire informazioni esterne al software che si occuperá del parsing del documento.

3.2 Flat milestone (CLIX)

Un altro approccio al problema dell'overlap é quello di CLIX (Canonical LMNL In XML), che é un metodo per codificare il data model LMNL all'interno di un documento XML. (le idee alla base di CLIX sono state proposte da Steve DeRose, Extreme Markup Language 2004, e si basano sui lavori legati allo sviluppo di OSI. Questa é stata la premessa per uno sviluppo piú organico di una rappresentazione canonica di LMNL in XML, per cui queste prime proposte si riflettono meglio in ECLIX che in CLIX).

Secondo le specifiche TEI, o in ECLIX, si ricorre all'uso di milestone per delimitare elementi della gararchia secondaria *solo se* producono una situazioni di overlap. Al contrario, l'approccio di **CLIX** esprime *ogni elemento* della gerarchia con una milestone, quelli primari come quelli secondari, gli elementi profani allo stesso modo di quelli sacri, senza considerare se le

milestone siano richieste per risolvere una situazine di overlap o meno.

Quando un documento CLIX viene interpretato come un albero XML risulta totalmente piatto (flat in inglese, da cui il nome): il documento é costituito da una sequenza di caratteri che si alternano ad elementi vuoti. L'unica eccezione é rappresentata ovviamente dal nodo radice del documento. Non esiste quindi alcuna gerarchia nel documento, e non é possibile determinare alcuna classificazioni di vocabolari o di situazioni di overlap senza l'uso di software specifico.

In CLIX é possibile rappresentare **self-overlapping**, servendosi di uno schemo di co-indexing, mentre non c'é alcun supporto per esprimere **virtual element**. La relazione di **dominanza** puó venire disaccoppiata da quella di **contenimento** solamente utilizzando informazioni esterne, allo stesso modo in cui si faceva con le milestone. L'unica differenza sta nel fatto che in questo caso la relazione di contenimento deve essere definita in termini di coppie di milestone, invece che di coppie di tag di inizio e di fine.

3.3 Fragmentation (o Partial Elements)

La frammentazione é un'altra tecnica per gestire situazioni di overlap: quando un elemento appartenente a un vocabolario secondario é in overlap con elementi appartenenti a quello primario, viene spezzato in frammenti piú piccoli (chiamati anche *partial element*), in numero sufficiente per risolvere la situazione di overlap. In questo modo si ottiene un documento in cui la gerarchia principale viene espressa attraverso markup XML standard, mentre gli elementi delle gerarchie secondarie risultano frammentati all'interno degli elementi principali, e connessi fra di loro attraverso attributi specifici.

Vengono utilizzate numerose convenzioni per distinguere fra partial e non-partial element, e per identificare i diversi frammenti dello stesso elemento. Questa tecnica é stata introdotta nelle TEI guidelines, in cui vengono descritte diverse varianti possibili:

1. **Virtual join**, **chaining** o **aggregation:** viene utilizzato il meccanismo di **co-indexing**. Ogni frammento viene identificato dall'attributo `@xml:id`, e puó essere collegato al successivo tramite l'attributo `@next`, al precedente tramite l'attributo `@prev`, oppure ad entrambi.
   Esempio: il problema in questo caso sta nel rappresentare i frammenti s, identificati con qs3 e qs4, come una singola (ma discontinua) unitá di questo testo:

```
<q>
     <s xml:id="qs2">Monsieur Paul, after he has taken equal
          parts of goose breast and the finest pork, and
          broken a certain number of egg yolks into them,
          and ground them <emph>very</emph>, very fine,
          cooks all with seasoning for some three hours.
     </s>
     <s xml:id="qs3">
          <emph>But</emph>,
     </s>
</q>
<s xml:id="ps2">she pushed her face nearer, and looked with
     ferocious gloating at the pâté
     inside me, her eyes like X rays,
</s>
<q>
     <s xml:id="qs4">he never stops stirring it!</s>
     <s xml:id="qs5">Figure to yourself the work of it —</s>
     <s xml:id="qs6">stir, stir, never stopping!</s>
</q>
```

(Fisher, M. F. K. I Was Really Very Hungry In As They Were, Knopf (1982), p. 43.)

É possibile collegare un frammento al successivo tramite l'attributo `@next`:

```
<s xml:id="qs3" next="#qs4"><emph>But</emph>,</s>
…
<s xml:id="qs4">he never stops stirring it!</s>
```

oppure, in maniera opposta, si puó collegare un frammento a a quello precedente usando l'attributo `@prev`:

```
<s xml:id="qs3"><emph>But</emph>,</s>
…
<s xml:id="qs4" prev="#qs3">he never stops stirring it!</s>
```

oppure ancora, effettuare un doppio link fra frammenti utilizzando entrambi gli attributi `@next` e `@prev`:

```
<s xml:id="qs3" next="#qs4"><emph>But</emph>,</s>
…
<s xml:id="qs4" prev="#qs3">he never stops stirring it!</s>
```

2. **Attributo @part:** per alcuni elementi (`<ab>`, `<l>`, `<lg>`, `<div>` e per elementi che appartengono alla classe `att.segLike`) esiste una semplificazione del meccanismo di virtual join presentato in precedenza: é possibile utilizzare un'istanza dell'elemento oppurtuno e specificare l'attributo `@part` con il valore I (Initial), M (Medial) o F (Final) (come descritto in TEI 16.3)

```
<s part="I"><emph>But</emph>,</s>
…
<s part="F">he never stops stirring it!</s>
```

3. **Join:** É possibile esprimere situazioni di virtual join in maniera equivalente in altri due modi:
   - tramite l'elemento `<link>`, con un attributo `@type` con valore “join” che specifica che il link deve essere interpretato come l'unione in un singolo aggragato degli elementi specificati in `@target`:

     ```
     <link type="join" targets="#qs3 #qs4"/>
     ```

   - tramite l'elemento `<join>`: in questo caso, a differenza dell'elemento `<link>`, é possibile anche specificare in aggiunta informazioni sul virtual element che rappresenta, attraverso l'attributo `@result`. In questo caso, a differenza dell'elemento `<link>`, la posizione all'interno del documento dell'elemento `<join>` é significativa: deve essere posto in una posizione in cui l'elemento indicato dall'attributo `@result` risulti contestualmente lecito:

     ```
     <join targets="#qs3 #qs4" result="s"/>
     ```

É importante notare come in questo caso per aggregare frammenati di elementi sovrapposti si sia ricorso all'uso di markup che non appartiene in maniera diretta al vocabolario usato per la codifica del dovumento. [*]

4. **Intermediate pointers:** é un'altra possibilitá equivalente ai virtual join per puntare a segmenti discontinui. Specificando piú di un elemento come valore dell'attributo `@target` degli elementi `<ptr>` o `<ref>`, gli elementi indicati vengono combinati o l'aggregati in

qualche maniera per produrre l'oggetto del puntatore. Tale aggragazione é comunque compito dell'applicazione che processa il documento, e non puó quindi essere definita semplicemente dal markup.

```
<ptr xml:id="point.qs3-qs4" target="#qs3 #qs4"/>
```

Come nel caso del join, anche in questa soluzione si ricorre all'uso di markup che non appartiene al vocabolario di codifica. [**]

5. **Attributo @n:** In maniera complementare alla prima, é possibile rappresentare elementi frammentati usando l'attributo globale `@n`: in questo modo é possibile mettere in relazione i frammenti costituenti un'unica unitá logica.

```
<s n="s-unit1"><emph>But</emph>,</s>
...
<s n="s-unit1">he never stops stirring it!</s>
```

Problemi comuni a tutti esempi fragmentation:

1. É necessario scegliere quale gerarchia debba prevalere sulle altre all'interno del documento XML. In alcuni casi, ad esempio per commenti o revisioni di testi, é ovvio che ci sia una gerarchia primaria (che si esprime nel markup generale del documento), mentre le altre rimangono in secondo piano. Ci sono invece altri casi in cui le gerarchie assumono la stessa importanza, per cui risulta poco appropriata o comunque arbitraria la scelta fra una gerarchia e l'altra.
2. Poiché un componente puó essere diviso in numerosi frammenti, spesso risulta difficile comprendere che queste parti costituiscano un'unitá unica. Questo comporta problemi di leggibilitá e, conseguentemente, di manutenibilitá
3. Contenimento => dominanza (solo con join e intermediate pointer si puó evitare)

	Self overlap	**Discontinuous element**	**Contenimento => dominanza**
Virtual join	SI grazie lo schema di puntatori	SI perché regione di testo fra due partial element non fa parte dell'elemento frammentato	SI L'uso di un singolo documento XML per codificare gerarchie differenti puó portare a generare relazioni di dominanza fra elementi profani che l'autore non aveva intenzione di esprimere
@part	NO: manca un meccanismo per distinguere frammenti appartenenti ad istanze diverse di elementi dello stesso tipo		SI
	NON SEMPRE Non é possibile tutti i casi di self overlap, né di occorenze annidate di elementi dello stesso tipo. Infatti, se gli elementi in questione sono divisi in frammenti, non é possibile identificare sempre quale frammento iniziale, di mezzo o finale debba essere combinato con gli altri costituenti lo stesso elemento, né in quale ordine ció debba essere fatto.	NON SEMPRE Per lo stesso motivo di self overlap	
Join	SI	SI	NO?
Intermediate pointers	COME JOIN?	COME JOIN?	COME JOIN?
@n	SI	SI	SI

3.4 Twin documents

Concettualmente, il metodo piú semplice per districare due (o piú) viste gerarchiche delle stesse informazioni che sono in conflitto fra di loro consiste nel codificarle due (o piú) volte, in modo che ognuna risulti catturare una singola vista.

Quando si parla di twin documents si intende quindi una collezione di documenti XML che condividono lo stesso markup sacro, intercalato da porzioni distinte di markup profano. Questo approccio é stato presentato in letteratura con nomi diversi, come distributed documents, multiple encodings of the same information, e redundant coding in multiple forms.

```
<TEI> <! -- verse vocabulary -->
   <teiHeader>...</teiHeader>
   <text>
      <body>
         <l>
            <speaker>Tibere</speaker>
            Poursuivez ...
            <speaker>Agrippine</speaker>
            Quoi , Seigneur ?
            <speaker>Tibere</speaker>
            Le propos detestable
         </l>
         <l>
            ou je vous ai surprise.
            <speaker>Agrippine</speaker>
            Ah ! Ce propos damnable
         </l>
         <l>
            d'une si grande horreur tous mes sens travailla
         </l>
      </body>
   </text>
</TEI>

<TEI> <!-- performance vocabulary -->
   <teiHeader>...</teiHeader>
   <text>
      <body>
         <sp>
            <speaker>Tibere</speaker>
            Poursuivez ...
         </sp>
         <sp>
            <speaker>Agrippine</speaker>
            Quoi , Seigneur ?
         </sp>
         <sp>
            <speaker>Tibere</speaker>
            Le propos detestable
            ou je vous ai surprise
         </sp>
```

```
            <sp>
               <speaker>Agrippine</speaker>
               Ah ! Ce propos damnable
               d'une si grande horreur tous mes sens travailla
            </sp>
         </body>
      </text>
   </TEI>
```

Attraverso twin documents ogni gerarchia risulta in un documento XML distinto, ognuno dei quali fa riferimeno alla propria struttura ad albero. Non é quindi necessaria alcuna distinzione fra gerarchia primaria e secondaria, né é richiesto alcuno strumento per esaminare ogni singola gerarchia separatamente. I principali **svantaggi** sono che é richiesto il mantenimento di copie multiple dello stesso markup sacro (un'invito quindi all'inconsistenza), e che non é presente alcuna indicazione esplicita su come queste viste multiple, che si trovano in file separati, siano collegate fra di loro. (Se il contenuto testuale delle annotazioni nei diversi documenti é lo stesso, é possibile utilizzare il testo stesso come mezzo per mettere in relazione le differenti annotazioni, come descritto in Witt 2002). Nonostante questo sia possibile, risulta comunque complicato gestire queste gerarchie, o accedere alle informazioni contenute in un documento mentre se ne sta processando un altro. Non é invece possibile osservare le ralzioni rfa gerarchie profane esaminando un singolo documento.

Inoltre, poiché gli elementi profani sono memorizzati in documenti distinti, non sempre é possibile stabilire un **ordinamento totale** fra tag di inizio e tag di fine di tali elementi. Ad esempio il primo elemento `<l>` (memorizzato nel primo documento) e il primo elemento `<sp>` (memorizzato nel secondo documento) si trovano entrambi fra gli elementi sacri `<body>` e `<speaker>`, e anche la loro posizione dall'inizio dei rispettivi documenti é la stessa (espressa in numero di byte). In situazioni come questa non é possibile affermare quale tag di inizio preceda l'altro. Se volessimo perció unire i due documenti in un singolo documento avremmo perció almeno due possibilitá. Questo significa anche che non vengono codificate gerarchie di **dominanza** indesiderate fra elementi profani di gerarchie differenti durante la serializzazione.

Questa tecnica non consente di rappresentare **virtual elements**, mentre situazioni di **self-overlap** possono essere codificate solo se i dizionari profani non sono disgiunti.

3.5 Stand-off markup (Out of Line markup)

Questa tecnica si basa sull'idea di esprimere piú gerarchie in overlap su uno stesso contenuto in un *documento sorgente* (che puó contenere XML o testo semplice) e uno o piú *documenti XML esterni*, ognuno con il proprio markup, che fanno riferimento al documento sorgente attraverso un qualche meccanismo di puntamento. Ogni documento esterno puó essere visto come una gerarchia indipendente (profana) sul documento sorgente che contiene tutto il markup sacro. Le TEI guidelines indicano come sia possibile mantenere stand-off markup all'interno dello stesso documento sorgente in apposite sezioni, invece di esternalizzarlo in altri documenti. Le specifiche TEI suggeriscono come trasformare (esternalizzare) un singolo documento XML di questo tipo in uno o piú documenti, di cui uno rappresenterá il documento sorgente, mentre gli altri saranno documenti esterni.

TEI suggerisce di utilizzare Xpointer per fare riferimento a nodi e caratteri del documento sorgente. Utilizzando per il puntamento un potente meccanismo come Xpointer, all'interno dei documenti esterni é possibile creare **discontinuous elements**. In situazioni standard in cui i vocabolari profani e sacri sono disgiunti, e in cui i documenti esterni possono fare riferimento solo al documento sorgente (quindi non al documento esterno di origine stesso) non é possibile esprimere **self-overlap**. In modo simile al caso di twin document, l'uso di documenti esterni evita la

creazione di relazioni di **dominanza** non desiderate fra nodi profani di gerarchie differenti.

Un altro problema é la difficoltá di mantenimento di documenti che fanno uso di stand-off markup: gli offset all'interno dei file possono cambiare in seguito anche a piccoli cambiamenti di editing dei documenti. Per risolvere questo problema é possibile aggiungere marker speciali all'interno dei documenti per fornire ID cui fare riferimenti col sistema di puntamento. Questa soluzione non é sicura in assoluto, in quanto non é da escludere che anche questi ID cambino - ad esempio, un gestore di un sito puó modificarli quotidianamente in maniera automatica, per evitare che dall'esterno venga fatto riferimento ai propri documenti.

4 Tabella riassuntiva

In ultima pagina é riportata una tabella riassuntiva per agevolare il confronto delle varie soluzioni presentate.

Bibliografia

[1]	DeRose. S. (2004). Markup overlap: A review and a horse. In Extreme Markup Languages.
[2]	Marinelli, P., Vitali, F., Zacchiroli, S. (2008). Towards the unification of formatsfor overlapping markup. The New Review of Hypermedia and Multimedia.
[3]	Sperberg-McQueen, C. M., Huitfeldt, C. (2008). Markup Discontinued:Discontinuity in TexMecs, Goddag structures, and rabbit/duck grammars.
[4]	Sperberg-McQueen, C. M., Burnard, L. (2005). TEI P5 Guidelines for Electronic Text Encoding and Interchange (revised). The Association for Computers and the umanities.
[5]	Tennison, J. (2008). Overlap, Containment and Dominance. Article from "Jeni's Musings" blog, disponibile all'indirizzo http://www.jenitennison.com/blog/node/95
[6]	Tennison, J. (2008). Representing Overlap in XML. Article from "Jeni's Musings" blog, disponibile all'indirizzo http://www.jenitennison.com/blog/node/97.
[7]	Cowan, J., Tennison, J., Piez, W., ECLIX: Reading XML as LMNL. LMNL wiki, disponibile all'indirizzo http://www.lmnl.org/wiki/index.php/ECLIX
[8]	Cowan, J., Tennison, J., Piez, W., CLIX Wiki, consultabile all'indirizzo http://www.lmnl.org/wiki/index.php/CLIX

(ALCUNE) NOTE - DUBBI

[*] In fondo Join (se non utilizza un altro meccanismo di puntamento, tipo XPointer) non é fragmentation + stand-off markup? Cioé Xpointer permette di evitare fragmentation, mentre usando join il documento va comunque frammentato, e poi i frammenti vengono riuniti puntando agli id dei frammenti. Peró i join vanno espressi in sezioni specifiche del documento, separate da dove viene espresso il vocabolario primario.

[**] Il principale vantaggio delle soluzioni presentati ai punti 3 e 4 é che tutte le gerarchie vengono espresse in maniera esplicita: quella primaria viene rappresentata in maniera diretta, allo stesso modo dele gerarchie alternative (secondarie), che sono state separate in frammenti e ricostruite via join.

	SELF OVERLAP	DISCONTINUOUS ELEMENT	CONTENIMENTO slegato da DOMINANZA		PROS	CONS
MILESTONE	**SI** (con co-indexing)	**NO** (*discontinuous elements* supportati aggiungendo altri metodi: next/prev o join - DeRose)	LIMITE DOCUMENTO UNICO	**NO** (si puó evitare, fornendo informazioni esterne al parser)	Manutenibilitá**:** il markup viene mantenu-to unito al contenuto	Processabilitá XML: richiesto apposito software che riconosca elementi e ricostruisca le struttura
					Vocabolario principale – vocabolari secondari	
					Facile vedere struttura principale	Difficile identificare contenuto delle strutture in overlap
FLAT MILESTONE	**SI** (con co-indexing)	**NO** (*discontinuous elements* supportati aggiungendo altri metodi: next/prev o join - DeRose)		**NO** (si puó evitare, fornendo informazioni esterne al parser) **NB:** *contenimento* definito in termini di coppie di milestone invece di coppie di start/end tag	Non é possibile determinare nessuna classificazione dei vocabolari o situazioni di overlap	
						É necessario software specifico
					Nessuna gerarchia sopravvive nel documento	
					Tutte le strutture sono trattate equamente	Difficile osservare le strutture
FRAGMENTATION	**SI** (con meccanismo di puntamento)	**SI** (supportati in maniera naturale con partial elements)		**NO**	Facile risolvere il contenuto delle strutture in overlap	
					Vocabolario principale – vocabolari secondari	
					Facile vedere struttura principale	Favorita la struttura principale, ma comunque abbastanza facile risolvere il contenuto delle strutture in overlap
TWIN DOCUMENTS	**NO** (se vocabolari sono disgiunti. Bisognerebbe avere numero di documenti variabili in funzione di quanto self-overlap é presente)	**NO**	COLLEZIONE DI DOCUMENTI	**SI**	Ogni gerarchia in documento XML separato con propria struttura ad albero	
					• Nessuna distinzione fra gerarchia princi-pale e secondaria • Facile riconoscere e gestire singole strutture	Copie multiple dello stesso markup sacro: • manutenibilitá • allineamento: difficile individuare relazioni fra gerarchie profane osservando un solo documento • non sempre possibile ordinamento totale fra elementi profani
STAND-OFF MARKUP	**NO** (nel per caso generale: • doc.esterni *possono puntare solo* a source document • vocabolari sacri e profani *disgiunti*)	**SI** (in generale dipende da meccanismo puntamento: con XPointer x es. é possibile)		**SI**	Markup sacro in source document, markup profano in external documents	
					Manutenibilitá: • modifica di external documents: gestione semplice • modifica source document: piú comples-sa perché é possibile rompere o modificare link. Meccanismo di IDRef puó aiutare, ma non risolve tutti i problemi	Difficile da processare con strumenti XML standard