Alma Mater Studiorum - Università di Bologna

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Specialistica in Informatica
Materia: Tecnologie Web

Risoluzione e linearizzazione di overlap in linguaggi di markup

Tesi di Laurea di
Francesco Poggi

Relatore
Chiar.mo Prof. **Fabio Vitali**
Correlatore
Dott. **Silvio Peroni**

Parole chiave:
EARMARK, Overlapping markup, Conversione di documenti, OWL, XML

Sessione I

Anno Accademico 2010-2011

Indice

Capitolo 1

INTRODUZIONE

In questa tesi viene proposto un meccanismo che permette di convertire documenti EARMARK in linguaggi di markup di tipo embedded. Questo lavoro si inserisce all'interno di un progetto di ricerca portato avanti dal dipartimento di Scienze dell'Informazione che si occupa di studiare i limiti dei principali linguaggi di markup. In particolare questa attività si concentra sul problema dell'overlapping markup [DUR02] [DO02], che rappresenta la principale questione ancora da risolvere nei linguaggi di markup tradizionali.

In questo contesto con il termine overlap facciamo riferimento a tutte quelle situazioni in cui elementi di markup multipli e indipendenti devono essere applicati allo stesso contenuto, andando a costituire strutture gerarchicamente incompatibili. I linguaggi di markup più diffusi, come ad esempio XML e tutti i linguaggi che derivano da SGML, sono in grado di gestire correttamente ognuna di queste singole strutture, ma presentano profondi limiti quando si tenta di farne coesistere più di una contemporaneamente all'interno di un solo documento. Il problema principale è dovuto al fatto che questa famiglia di linguaggi di markup si basa su un modello ad al-

bero che impone una rigida organizzazione gerarchica che non consente di annotare un elemento con descrittori che non si annidano correttamente.

Per risolvere questo problema esistono molte soluzioni: alcune intervengono a livello di linguaggio di markup, introducendo particolari tecniche di codifica, altre a livello di linguaggio di meta-markup, estendendo soluzioni già esistenti aggiungendo costrutti specifici, altre ancora a livello di modello astratto, sostituendo lo schema ad albero con una struttura più generale. Al dipartimento è stato sviluppato EARMARK (Extremely Annotational RDF Markup) [DPV11a][DPV11b], un meta-linguaggio di markup che suggerisce un nuovo approccio al meta-markup basato su tecnologie del Semantic Web. Il principale vantaggio di EARMARK è che propone una soluzione che permette di integrare i vantaggi dell'approccio al markup dei formati embedded (come XML) a quelli delle annotazioni esterne (come RDF) in un unico modello completo. Le caratteristiche di EARMARK consentono di rappresentare in maniera esplicita caratteristiche documentali complesse (strutture in overlap, contenuto ripetuto, elementi discontinui, attributi strutturati, ecc.) che sono difficilmente esprimibili nei linguaggi di markup tradizionali.

Questo progetto prevede di sviluppare un insieme di tecnologie e di strumenti simili a quelli disponibili per XML che forniscano supporto allo sviluppo di applicazioni basate su EARMARK. In particolare è stato previsto lo sviluppo di un framework che consenta di mettere in rapporto EARMARK con i principali linguaggi di markup di tipo embedded. L'obiettivo è quello di realizzare un framework che permetta di convertire documenti EARMARK in documenti XML, e viceversa. Il processo di trasformazione realizzato dal framework EARMARK è diviso in due fasi: ROCCO (Read, Overhaul, Convert, Classify, Organize), che si occupa di eseguire la trasformazione di file XML in documenti EARMARK, e FRETTA (From EARMARK To Tag), che esegue l'operazione inversa, convertendo documenti EARMARK nel formato XML. Questa tesi propone un'implementazione di FRETTA, un motore che realizza la seconda fase di questa conversione.

Il problema principale affrontato durante lo sviluppo di FRETTA riguarda la gestione delle strutture in overlap: il modello ad albero su cui si basa XML non permette infatti di esprimere in maniera esplicita più gerarchie all'interno dello stesso documento. Il principale punto di forza di questo approccio è rappresentato dai vantaggi pratici che fornisce in termini di semplicità e funzionalità nei processi di

creazione, modifica e formattazione dei testi: mentre i modelli alternativi, come ad esempio i grafi diretti, portano a soluzioni pratiche spesso inefficaci ed inadeguate, il modello ad albero supporta in maniera semplice ed efficiente le principali operazioni necessarie per eseguire l'elaborazione di documenti. La caratteristica di potere esprimere una sola visione del contenuto di un documento non era considerato un aspetto limitante di questo approccio: per lungo tempo il problema dell'overlapping markup non ha infatti suscitato molto interesse in quanto si è pensato che fosse legato alla complessità intrinseca di contesti applicativi molto specifici. Nel campo delle discipline umanistiche, e in particolare dell'analisi linguistica e letteraria, la possibilità di descrivere e analizzare l'overlap fra le varie strutture dei testi (ad esempio strutture poetiche e sintattiche) permette a studiosi e accademici di acquisire una conoscenza più profonda dei testi stessi.

Esistono però molte situazioni comuni in cui il problema dell'overlap si dimostra essere una questione determinante: ad esempio tutti gli ambiti in cui più autori sono coinvolti in progetti che riguardano insiemi di documenti, oppure i contesti in cui, lavorando con documenti che vengono modificati nel tempo, si ha la precisa intenzione di tenere traccia di modifiche successive. In ambito legislativo è comune fare riferimento ad altri testi e codici: le nuove leggi ad esempio emendano spesso altri documenti prodotti in precedenza (si veda ad esempio il portale http://www.legislation.gov.uk/changes). Un altro caso è quello degli strumenti di editing che implementano la funzionalità di change tracking [WB11] [ISO08]: la rappresentazione di più gerarchie (le diverse versioni del testo) in un unico documento richiede infatti la gestione di strutture in overlap sullo stesso contenuto.

Le principali soluzioni proposte suggeriscono di affrontare il problema a livelli diversi: alcune lavorano a livello di linguaggio di markup, introducendo particolari tecniche di codifica che permettono di descrivere le strutture non direttamente esprimibili dai modelli gerarchici ad albero (per esempio milestone, fragmentation, ecc.) [BPS08] [SB90]; altre suggeriscono invece di intervenire a livello di linguaggio di meta-markup (ad esempio estendendo meta-linguaggi già esistenti, come nel caso di TexMECS [HS01], oppure introducendo all'interno del documento la possibilità di organizzare il contenuto in diversi layer, come in XCONCUR [SW06] e LMNL [TP02]); infine un'altra proposta è di seguire un approccio più radicale, intervenendo a livello di modello astratto e andando a sostituire la struttura ad albero con uno

schema più generale (ad esempio GODDAG [SH00] utilizza grafi diretti aciclici). FRETTA si occupa principalmente di gestire le soluzioni che esprimono le situazioni di overlap con specifiche tecniche di codifica, in quanto è l'approccio principale seguito da XML.

Per risolvere i problemi che derivano dall'imposizione di una struttura ad albero sono state proposte numerose tecniche che suggeriscono di identificare una gerarchia come primaria ed esprimerla direttamente all'interno del documento, e di nascondere le informazioni che riguardano le gerarchie secondarie in un'altra forma: elementi vuoti che indicano i limiti di un elemento, elementi divisi in frammenti, riferimenti indiretti, versioni multiple dello stesso documento, ecc (vedi sezione 2.3.1). La necessità di rispettare la gerarchia principale induce a ridurre ed offuscare l'importanza delle strutture secondarie e, di conseguenza, queste ultime risultano più difficili da identificare. Inoltre anche la gestione e l'elaborazione diventa più complicata: banali ricerche che potrebbero essere specificate con un breve Xpath [BBC10] spesso richiedono query molto lunghe e complesse, oppure per ottenere semplici visualizzazioni è necessario ricorrere a lunghi e intricati fogli di stile. In alcune situazioni è anche possibile che la struttura principale vincoli quella secondaria a tal punto da impedire di esprimere alcune caratteristiche delle gerarchie secondarie. Per questo la scelta di risolvere il problema dell'overlap con queste tecniche rende il documento più difficile da interpretare e meno intelleggibile, impedendo di sfruttare le potenzialità degli strumenti forniti da XML.

La soluzione fornita da EARMARK propone invece un diverso approccio al markup e alla gestione del problema dell'overlap: oltre fornire un modello che permette di gestire le caratteristiche di documenti basati su alberi come XML e altri meta-linguaggi di markup gerarchici, EARMARK consente di esprimere in maniera esplicita strutture in overlap senza la necessità di utilizzare specifiche tecniche di codifica. Durante lo sviluppo di EARMARK gli autori si sono infatti concentrati sulle modalità da utilizzare per memorizzare le strutture in overlap: l'obiettivo principale è stato quello di trovare una soluzione che permettesse di esprimere le strutture in overlap in maniera esplicita, in modo che risultino trasparenti e facilmente comprensibili sia agli utenti che alle applicazioni. Per raggiungere questo scopo EARMARK abbandona l'approccio embedded seguito dai linguaggi di markup più diffusi, come ad esempio XML e tutti gli altri linguaggi SGML-based, in favore

di una soluzione di tipo stand-off: i markup descriptor che costituiscono le strutture da descrivere non contengono direttamente le parti di documento o le sottostrutture cui si riferiscono, ma puntano ad esse attraverso altri meccanismi. L'idea di base è dunque quella di modellare un documento EARMARK come una collezione di frammenti di testo indirizzabili, e di associare a tale contenuto testuale delle asserzioni OWL che descrivano sia caratteristiche strutturali che proprietà semantiche legate a quel contenuto. In questo modo EARMARK è in grado di descrivere, oltre a casi semplici di documenti con una singola gerarchia, anche situazioni di gerarchie multiple in cui il contenuto testuale legato agli elementi del markup appartiene ad alcune gerarchie ma non ad altre. In questo modo tutte le caratteristiche documentali complesse legate al problema dell'overlap (simple overlap, self overlap, virtual element, discontinuous element, ecc.) possono essere gestite esplicitamente in EARMARK. Allo stesso modo è possibile aggiungere in qualsiasi momento annotazioni semantiche attraverso ulteriori asserzioni, e anche queste possono essere in overlap con il contenuto precedente (vedi sezione 3.1).

Un vantaggio evidente di EARMARK è la possibilità di accedere e navigare i documenti utilizzando strumenti molto diffusi e ampiamente supportati legati al Semantic Web. I documenti EARMARK sono infatti modellati attraverso ontologie OWL [HKP09], e le asserzioni che costituiscono i documenti sono semplici asserzioni RDF [MM04]. Di conseguenza è possibile utilizzare linguaggi di query come SPARQL [PS08] e un vasto insieme di strumenti esistenti per gestire strutture in overlap anche molto complicate: operazioni di ricerca e manipolazione di strutture arbitrariamente complesse che sono molto difficili o addirittura impossibili con le tecnologie correlate ad XML risultano semplici ed immediate all'interno del modello di EARMARK.

Il framework EARMARK propone quindi un meccanismo che consente di mettere in relazione EARMARK con XML. In particolare l’obiettivo di questa tesi è lo sviluppo di FRETTA, che è quella parte del framework che si occupa di convertire documenti EARMARK in documenti XML. Come detto in precedenza, il principale problema affrontato è legato alla conversione di quel sottoinsieme di caratteristiche di EARMARK legate al problema dell’overlap che non sono direttamente linearizzabili in XML. Da un punto di vista concettuale, FRETTA scompone il processo di trasformazione in due fasi separate che permettono di affrontare gli

aspetti sintattici e quelli semantici legati al formato oggetto della conversione in maniera separata:

1. per prima cosa FRETTA effettua una prima modifica alla struttura del documento che si occupa di convertire i costrutti EARMARK non direttamente esprimibili in XML secondo le tecniche di codifica definite dal formato di arrivo. In questa fase sarà compito dell'utente indicare a FRETTA le tecniche da utilizzare per convertire ogni singola situazione di overlap.
2. in secondo luogo viene modificata la struttura del documento in funzione della semantica dei formati di arrivo. Poniamo ad esempio che si intenda convertire un documento EARMARK contenente informazioni di change tracking nel formato OASIS Open Document [WB11]: in questo caso il framework deve interpretare secondo una certa semantica (es: change tracking) relativa a un certo formato (es: ODT) specifiche parti del documento EARMARK (sia elementi strutturali del documento che metadati con le informazioni di modifica ad essi relativi), e quindi eseguire le modifiche definite per quella particolare interpretazione (ad esempio iniettando alcuni di questi metadati all'interno della struttura del documento in specifici elementi di markup).

Attraverso questi due passi il framework fornisce un meccanismo generale per la conversione di documenti EARMARK in formati XML.

Il meccanismo di conversione di FRETTA si articola in quattro passi (specifica delle tecniche di codifica, conversione strutturale, conversione semantica, linearizzazione), ognuno dei quali prende in input un documento, compie su di esso una o più operazioni, producendo in output una nuova versione modificata del documento. Le operazioni di conversione svolte dai diversi passi vengono svolte su documenti EARMARK in quanto, come detto in precedenza, questo formato permette di gestire in maniera esplicita le feature complesse legate alle situazioni di overlap. La linearizzazione nel formato di destinazione viene realizzata solamente all'ultimo passo di FRETTA, dopo che il documento è stato convertito in modo da rispettare i vincoli del formato di arrivo.

I primi due passi della conversione effettuano una modifica strutturale al documento in modo che rispetti la sintassi del formato di arrivo: le indicazioni specificate dall'utente durante il primo passo impongono di adattare specifici

elementi ed attributi per esprimere la semantica dell'overlap in maniera implicita, alterando quindi la struttura gerarchica del documento. Nella seconda fase FRETTA effettua un'analisi del documento, ne modifica la struttura come indicato al passo precedente e genera una nuova versione del documento. FRETTA implementa sei tecniche di codifica: milestone, fragmentation, stand-off markup, copyOf, sameAs, dominance (vedi sez. 4.1 e 4.2).

Nella terza fase FRETTA si occupa di interpretare secondo una certa semantica relativa a un certo formato specifiche parti del documento EARMARK, e quindi eseguire le modifiche definite per quella particolare interpretazione: se in questo passo l'utente specifica di utilizzare la semantica relativa ad un particolare formato (al momento è stata implementata la semantica del formato TEI [SB05]), FRETTA utilizzerà tali semantiche per interpretare specifiche parti del documento EARMARK (sia elementi strutturali del documento che metadati ad essi relativi), e quindi eseguire le modifiche che l'utente ha indicato per quella particolare interpretazione (ad esempio iniettando alcuni di questi metadati all'interno della struttura del documento in specifici attributi ed elementi di markup). Per maggiori informazioni su questa fase consultare la sezione 4.3.

L’ultimo passo si occupa di effettuare la linearizzazione del documento nel formato di arrivo: in questo passo FRETTA prende in input un documento EARMARK e ne produce una versione linearizzata in XML. Il documento in input può essere quello generato nel passo II (EARMARK linearizzabile) o nel passo III (EARMARK semantico): l'utente infatti può decidere se eseguire solo la conversione strutturale, che consente di convertire le parti di documento non esprimibili in XML in strutture linearizzabili, oppure svolgere un ulteriore passo che interpreti e converta parti di documento secondo la semantica di uno specifico formato (vedi sez. 4.4).

Il risultato di questa fase è il prodotto finale di tutto il processo di conversione eseguito dal framework, e dovrebbe risultare in una versione XML in buona forma del documento in input. In questo passo però non viene eseguita alcuna forma di controllo sul documento: l'utente è infatti responsabile di specificare come deve essere eseguita la conversione, e i passi precedenti di FRETTA devono garantire la corretta esecuzione delle modifiche specificate dall'utente. Questo non assicura quindi che la conversione abbia successo: è compito dell'utente individuare le parti di documento non esprimibili in XML e specificare al framework le corrette

informazioni di gestione.

FRETTA è dunque un'applicazione completa che permette di convertire documenti EARMARK contenenti strutture di markup in overlap in formati XML. Il meccanismo realizzato è completamente configurabile: l'utente può dare infatti precise informazioni su come intende gestire ogni singola parte del documento da convertire. Infine l'architettura di FRETTA è estendibile: la versione attualmente implementata permette di gestire le situazioni di overlap con le tecniche di codifica di TEI P5, ma è possibile aggiungere in un secondo tempo nuove funzionalità che consentano la linearizzazione di documenti EARMARK in ulteriori formati XML. Il principale progetto di sviluppo di FRETTA è appunto legato all'implementazione di specifiche estensioni che permettano di effettuare la conversione fra i due principali formati di change tracking, ossia *OASIS Open Document* [WB11] utilizzato dalla suite *OpenOffice.org* e *Office Open XML* [ISO08] sviluppato da *Microsoft*.

Capitolo 2

OVERLAPPING MARKUP

L'oggetto principale di questa tesi è il markup, che rappresenta uno degli elementi chiave del processo di codifica dei testi in formati accessibili alle macchine. Possiamo definire come markup ogni mezzo per rendere esplicita una particolare interpretazione di un testo [CRD87]. Esistono vari tipi di markup, alcuni connaturati con il processo di scrittura (ad esempio markup puntuazionale e presentazionale), altri introdotti da tipi diversi di codifica (ad esempio nell'ambito della codifica digitale è presente il markup procedurale, descrittivo, referenziale, ecc.): il markup non è quindi soltanto un risultato di un processo di informatizzazione, ma qualcosa che esiste in varie forme anche prima dell'avvento dell'informatica. L'esempio più semplice è quello del markup puntuazionale: da millenni a questa parte quando un autore scrive usa un insieme prefissato di segni al fine di fornire informazioni sintattiche sul testo. Le parole sono separate da delimitatori di parola (gli spazi), le frasi sono separate da delimitatori di frase (le virgole) e i periodi sono separati dai delimitatori di periodo (i punti). E' evidente che questi segni non facciano

strettamente parte del contenuto del testo quanto del markup ad esso legato: durante il processo di fruizione di un testo nessuno dirà ad alta voce 'virgola', ma adotterà specifici comportamenti (ad esempio pause e intonazioni) per migliorare la comprensione del testo. Possiamo dire in questo caso che il markup rappresenta quelle aggiunte al testo scritto che permettono di renderlo più facilmente fruibile.

Limitare però il concetto di markup semplicemente ad applicazioni come la precedente in un contesto di codifica digitale dei testi sarebbe riduttivo. Con l'avvento dei sistemi di elaborazione di testi sono stati sviluppati nuovi tipi di markup e nuovi modelli di processing: quando un testo viene memorizzato in formato elettronico può essere annotato, oltre che con il markup tradizionale, anche con tipi particolari di markup progettati per l'elaborazione del testo stesso da parte di specifiche applicazioni elettroniche. E' possibile ad esempio utilizzare markup di tipo procedurale per indicare specifiche istruzioni che un'applicazione deve eseguire (ad esempio saltare una riga), markup descrittivo per identificare il tipo di entità del frammento corrente (ad esempio indicando che si tratta di un paragrafo), markup referenziale per rendere esplicito un collegamento a un'entità esterna al testo corrente, meta-markup per definire e controllare l'elaborazione di altre forme di markup.

Questa tesi si concentra sui linguaggi di markup di tipo descrittivo, che prevedono di identificare il tipo di ogni elemento del contenuto: invece di indicare comportamenti specifici, come gli aspetti grafici di allineamento e interlinea, l'approccio descrittivo propone di individuare il ruolo degli elementi del testo all'interno del documento, specificando ad esempio che un elemento è un titolo, un paragrafo, una citazione, ecc. Il markup descrittivo permette quindi di rendere esplicite le diverse strutture utilizzate dall'autore durante il processo di scrittura che si stratificano sul contenuto di un testo. Pensiamo ad esempio un autore di una poesia: oltre ad un piano strettamente sintattico, un poeta può fare uso di una specifica metrica, di figure retoriche, ecc. Un processo di codifica efficace dovrebbe dare la possibilità di mettere in luce queste diverse prospettive, in modo da rendere esplicita ogni singola interpretazione.

I linguaggi di markup più diffusi come ad esempio XML e tutti i linguaggi che derivano da SGML sono in grado di gestire correttamente ognuna di queste singole strutture, ma presentano profondi limiti quando si tenta di farne coesisterne più di

una contemporaneamente all'interno di un solo documento. Il problema principale è dovuto al fatto che questa famiglia di linguaggi si basa su un modello ad albero che impone una rigida organizzazione gerarchica che non consente di annotare un elemento con descrittori che non si annidino correttamente: tale questione è conosciuta come “problema dell’overlapping markup”.

Nei prossimi paragrafi viene presentato un breve excursus sullo stato dell'arte dei principali approcci al problema dell'overlapping markup, cercando di analizzare sia i limiti evidenziati da queste approcci, sia i punti di forza che sono stati di stimolo alla realizzazione della soluzione discussa in questa dissertazione.

Per prima cosa viene presentato il modello ad albero, che rappresenta la struttura dati utilizzata dai linguaggi di markup più diffusi (sez. 2.1), e viene introdotto il problema dell'overlapping markup, che costituisce la principale questione aperta legata al processo di codifica digitale di testi (sez. 2.2). In seguito vengono descritti i principali approcci presentati in letteratura a questo problema e vengono discusse le limitazioni più rilevanti di queste soluzioni (sez. 2.3). Il capitolo si conclude con la descrizione di un diverso approccio al problema dell’overlap che propone la realizzazione di un framework di conversione fra i principali formati di overlapping markup embedded (sez. 4.4).

2.1 OHCO thesis e modello ad albero

I linguaggi di markup più affermati, come XML e i linguaggi derivati da SGML[1], richiedono che le caratteristiche definite nella codifica di un documento siano organizzate gerarchicamente in una struttura ad albero, in cui ogni frammento di contenuto del documento è contenuto in un unico elemento di markup, ed ognuno di questi elementi è a sua volta contenuto in un solo elemento genitore, e così via fino ad un singolo elemento che, al vertice di questa gerarchia, rappresenta la radice del documento. Questo modello deriva da un'idea, conosciuta come OHCO thesis [DDM90], che sostiene che il migliore modo di rappresentare un testo sia attraverso

[1] In realtà SGML fornisce una caratteristica opzionale chiamata CONCUR che permette di rappresentare alcune situazioni di overlap. Dove non specificato diversamente, quando parleremo di SGML faremo però riferimento alla versione standard di SGML con l’opzione CONCUR impostata a “false”.

una “gerarchia ordinata di oggetti” (Ordered Hierarchy of Content Objects): questa filosofia sostiene appunto che i testi siano strutturati essenzialmente in oggetti (ad esempio capitoli, sezioni, paragrafi, liste, ecc..) annidati gerarchicamente gli uni dentro gli altri [CRD87].

La visione OHCO si è dimostrato un modello molto efficace che ha sostenuto l'approccio del markup descrittivo nel processo di rappresentazione di testi letterari in formato elettronico e, più specificamente, ha fornito un framework adatto per la Text Encoding Initiative (TEI, [SB05]) prima, e per XML dopo. A sostegno della rilevanza di questa tesi è interessante segnalare che anche prima che fosse presentata per questo scopo, il punto di vista del testo come gerarchia di content object era già utilizzato in maniera implicita durante lo sviluppo dei primi software di composizione ed elaborazione di testi [GOL81], [REI80].

Il principale punto di forza di questo approccio è rappresentato dai vantaggi pratici che fornisce in termini di semplicità e funzionalità nei processi di creazione, modifica e formattazione dei testi: mentre i modelli alternativi, come ad esempio i grafi diretti, portano a soluzioni pratiche inefficaci ed inadeguate, l'approccio OHCO e il particolare modello ad albero che esso impone supporta in maniera semplice ed efficiente le principali operazioni necessarie per eseguire l'elaborazione di documenti.

E' interessante notare come questa tesi abbia subito solo piccoli raffinamenti, estensioni e chiarimenti negli anni successivi e che, di fatto, sia stata adottata come base teorica nel campo della ricerca sull'elaborazione dei testi e sulla standardizzazione dei formati di codifica. Solo all'inizio degli anni novanta si è cominciato a metterla in discussione in seguito ai problemi pratici segnalati dai professionisti coinvolti nel processo di codifica dei testi. Il principale di questi è noto come “problema dell'overlapping markup”, che è appunto l'argomento principale oggetto di questa dissertazione.

2.2 Overlapping markup

La questione dell'overlap è certamente il problema più rilevante che deve

essere risolto dai linguaggi di markup. In questo contesto con il termine overlap facciamo riferimento a tutte quelle situazioni in cui elementi di markup multipli e indipendenti devono essere applicati allo stesso contenuto, andando a costituire strutture gerarchicamente incompatibili. Per chiarire questo concetto viene fornito un esempio concreto di un documento che presenta situazioni di overlap. L'esempio è un estratto della tragedia *La Mort d'Agrippine* di Cyrano de Bergerac [CM95], e presenta due strutture indipendenti: la struttura dei versi organizzati in righe con una data metrica e rima, e la struttura dei versi costituita dalle battute recitate dai personaggi. Si consideri il seguente frammento:

Tibère
Poursuivez...
Agrippine
Quoi, Seigneur?
Tibère
Le propos détestable
Où je vous ai surprise.
Agrippine
Ah! Ce propos damnable
d'une si grande horreur tous mes sens travailla

Esempio 2.1 un testo in cui più strutture sono in overlap sullo stesso contenuto

```
<TEI>
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <l>
        <sp><speaker>Tibère</speaker>Poursuivez...</sp>
        <sp><speaker>Agrippine</speaker>Quoi, Seigneur?</sp>
        <sp><speaker>Tibère</speaker>Le propos détestable
      </l>
      <l>
        où je vous ai surprise.</sp>
        <sp><speaker>Agrippine</speaker>Ah! Ce propos damnable
      </l>
      <l>
        d'une si grande horreur tous mes sens travailla</sp>
      </l>
    </body>
  </text>
</TEI>
```

Esempio 2.2 una codifica XML TEI non in buona forma del testo dell'esempio 2.1.

La codifica dell'esempio 2.1 proposta contiene due casi di overlap: un discorso

inizia in una riga (linea 8) e finisce in un'altra riga (linea 11), e una situazione analoga si verifica da linea 12 a 15. Per questi motivi l'esempio 2.2 non è un documento XML ben formato, per cui questo documento verrebbe rifiutato da un qualsiasi strumento conforme alla specifica XML. Un altro aspetto interessante che emerge dall'esempio precedente è che, anche se alcuni elementi sono si sovrappongono, altri si annidano correttamente: ad esempio gli elementi al vertice della gerarchia sono condivisi sia dalla struttura dei versi che da quella delle battute, per cui non vanno in overlap.

Per lungo tempo il problema dell'overlapping markup non ha suscitato molto interesse in quanto si è pensato che fosse legato ad ambiti di interesse molto ristretti: nel campo delle discipline umanistiche ad esempio, e in particolare dell'analisi linguistica e letteraria, la possibilità di descrivere e analizzare l'overlap fra le varie strutture dei testi (ad esempio strutture poetiche e sintattiche) permette a studiosi e accademici di acquisire una conoscenza più profonda dei testi stessi. La complessità delle soluzioni proposte non è mai stata presa in seria considerazione in quanto si pensava fosse da imputare alla complessità intrinseca dei contesti stessi.

Esistono però molti esempi di situazioni comuni in cui il problema dell'overlap si dimostra essere una questione determinante: ad esempio tutti gli ambiti in cui più autori sono coinvolti in progetti che riguardano insiemi di documenti, oppure i contesti in cui, lavorando con documenti che vengono modificati nel tempo, si ha la precisa intenzione di tenere traccia di modifiche successive. In ambito legislativo ad esempio è comune fare riferimento ad altri testi e codici: le nuove leggi ad esempio emendano spesso altri documenti prodotti in precedenza. Gli autori della legislazione di solito non sono però interessati a rispettare la struttura dei documenti originali, e gli emendamenti spesso si sovrappongono alle gerarchie in cui sono organizzati i documenti cui fanno riferimento, creando sovrapposizioni sulle strutture in cui questi sono organizzati (ad esempio parti di paragrafi o di liste)**:** questo problema è stato affrontato in maniera approfondita durante lo sviluppo del portale *legislation.gov.uk*, un sito del governo britannico che fornisce un servizio specifico che permette di consultare una lista dettagliata delle modifiche fatte da tutte le leggi promulgate dal 2002[2]**.** Un altro esempio che richiede la gestione di strutture in overlap è quello degli strumenti di editing che implementano la funzionalità di change tracking. Due

[2] http://www.legislation.gov.uk/changes

formati molto diffusi che permettono di mantenere informazioni relative alle modifiche successive apportate ai documenti sono *OASIS Open Document* [WB11], utilizzato dalla suite *OpenOffice.org*, e *Office Open XML* [ISO08], sviluppato da *Microsoft*. Anche in operazioni apparentemente molto semplici, come nel caso in cui due paragrafi distinti vengano accorpati in un unico paragrafo, emergono con evidenza elementi di complessità di non semplice gestione: i markup descriptor che rappresentano i paragrafi e quelli che descrivono l'operazione di cancellazione fanno riferimento alla stessa porzione di contenuto, per cui l'elemento che fa riferimento alla cancellazione contiene la fine del primo paragrafo e l'inizio del successivo, e di conseguenza la gerarchia di contenimento viene violata generando una situazione di overlap.

2.2.1 Caratteristiche complesse legate all'overlap

Alcuni testi presentano caratteristiche complesse che difficilmente si adattano al modello ad albero utilizzato dai principali linguaggi di markup come XML. E' possibile riassumere gli aspetti principali come segue:

1. **simple overlap:** è il caso in cui si intende annotare lo stesso frammento di documento con due markup descriptor diversi, ossia con elementi con diverso general identifier. Tipici esempi sono gli scenari in cui è necessario identificare gerarchie multiple concorrenti sullo stesso documento (ad esempio strutture grammaticali, metriche, fonetiche, tipografiche, ecc.). Quando si tenta di codificare più gerarchie in un singolo documento è probabile che una struttura si sovrapponga ad un´altra: ad esempio una strofa iniziata in una pagina può terminare in quella successiva.
2. **self overlap:** in questo caso a sovrapporsi sono due elementi della stessa struttura e che hanno lo stesso general identifier. Ad esempio consideriamo un documento di testo che deve essere commentato da due diversi revisori. Supponiamo che questi revisori vogliano annotare due regioni di testo parzialmente sovrapposte. È ragionevole considerare questi due commenti come appartenenti alla stessa struttura (la struttura dei commenti): il termine *"self" overlap* viene utilizzato per riferirsi a queste situazioni, poiché generano elementi con gli stessi nomi che si

sovrappongono gli uni agli altri. Un problema comune che deriva da gestioni banali di questi casi è che risulta impossibile distinguere le situazioni di self-overlap dagli annidamenti corretti degli elementi.

3. **virtual elements:** si tratta di elementi che non sono presenti in maniera esplicita all´interno del testo, ma la cui presenza può essere inferita da un´applicazione in funzione di una specifica tecnica di codifica [SB05]. I *virtual elements* vengono utilizzati per definire elementi il cui contenuto è il risultato della riorganizzazione di materiale presente altrove all´interno del documento. Un caso particolare di virtual elements è quello dei *discontinuous elements*, che sono elementi il cui contenuto non è una regione di testo contigua.

4. **Disaccoppiamento di contenimento e dominanza:** per contenimento e dominanza si fa riferimento ai concetti definiti da Sperberg-McQueen e Huitfeldt in [SH08]. La *dominanza* è la chiusura transitiva della relazione parent/child, per cui una parte di documento domina un´altra se è un suo antenato nella struttura del documento. Il *contenimento* è definito come la relazione superset/subset sui nodi foglia raggiungibili da un nodo seguendo gli archi nel verso parent/child, per cui una parte di documento contiene un´altra se contiene tutti i caratteri dell´altra parte.

   I linguaggi di markup basati su una struttura ad albero come XML tendono ad avere la proprietà che il contenimento implichi la dominanza, mentre spesso questo aspetto non viene richiesto. Se pensiamo a gerarchie multiple che condividono lo stesso contenuto di caratteri, non è evidente il motivo per cui porzioni di documento appartenenti a una gerarchia debbano essere messe in relazione via dominanza con parti strutturali appartenenti ad altre gerarchie.

2.3 Approcci attuali al problema dell'overlap

Uno dei principali assunti della letteratura sull'overlapping markup è che la filosofia di organizzare i documenti in strutture ad albero (come in SGML e XML) sia una semplificazione troppo limitante che necessita correzioni o estensioni.

Le principali soluzioni proposte suggeriscono di affrontare il problema a livelli

diversi: alcune lavorano a livello di linguaggio di markup, introducendo tecniche di codifica che permettono di descrivere le strutture non direttamente esprimibili dai modelli gerarchici ad albero (per esempio milestone, fragmentation, ecc.) [BPS08], [SB90]; altre suggeriscono invece di intervenire a livello di linguaggio di meta-markup (ad esempio estendendo meta-linguaggi già esistenti, come nel caso di TexMECS [HS01], oppure introducendo all'interno del documento la possibilità di organizzare il contenuto in diversi layer, come in XCONCUR [SW06] e LMNL [TP02]); infine un'altra proposta è di seguire un approccio più radicale, intervenendo a livello di modello astratto e andando a sostituire la struttura ad albero con uno schema più generale (ad esempio GODDAG [SH00] utilizza grafi diretti aciclici).

In questa sezione confronteremo le principali soluzioni che utilizzano questi approcci, fornendo per ognuna una possibile codifica del frammento presentato nell'esempio 2.1. In conclusione termineremo questa analisi confrontando i principali limiti delle diverse proposte.

2.3.1 Soluzioni a livello di linguaggio di markup

Un primo approccio al problema delle strutture in overlap è quello di affrontare la questione a livello di linguaggio di markup: questa proposta tenta di trovare una mediazione fra il bisogno di supportare le situazioni di overlap e la necessità di mantenere un'organizzazione strettamente gerarchica del markup dei documenti. L'intuizione comune a questo tipo di soluzioni è quella di identificare una struttura come primaria e di esprimerla in maniera standard con una singola gerarchia, e di rappresentare le altre in un altro modo (elementi vuoti che individuano i limiti di un elemento, elementi divisi in frammenti, riferimenti indiretti, ecc.), in maniera da non violare i vincoli del modello ad albero. Le cinque principali tecniche di gestione dell'overlap descritte in letteratura possono essere riassunte come segue [DER04], [TEN08]:

- *milestone*, in cui una gerarchia viene espressa utilizzando il modello standard, mentre gli elementi appartenenti alle altre vengono rappresentati attraverso un paio di elementi vuoti che indicano i punti di inizio e di fine dell'elemento, connessi gli uni agli altri da speciali attributi (vedi sez. 2.3.1.1).

- *flat milestone*, che rappresenta tutti gli elementi delle diverse gerarchie con milestone, ossia una coppia di elementi vuoti che indicano le posizioni di inizio e fine del markup descriptor, tutti contenuti come figli dello stesso elemento radice (vedi sez. 2.3.1.2).
- *fragmentation* (o *partial elements*), in cui la gerarchia primaria è espressa con il markup gerarchico standard, e gli elementi delle altre vengono frammentati all'interno degli elementi primari in modo da rispettare la gerarchia principale; gli elementi che rappresentano i frammenti dello stesso elemento vengono connessi attraverso un sistema di linking implementato da specifici attributi (vedi sez. 2.3.1.3).
- *twin documents*, in cui ogni gerarchia è rappresentata in un documento diverso, ognuno con lo stesso contenuto testuale ma annotato con i markup descriptor specifici di ogni singola gerarchia (vedi sez. 2.3.1.4).
- *standoff markup*, che pone tutto il contenuto testuale ed eventualmente una gerarchia condivisa in una singola struttura, e definisce gli elementi rimanenti in altre strutture (per es. file) o in posizioni specifiche del documento. Questi ultimi elementi sono collegati tramite un meccanismo di linking (per es. Xpointer [DDG02]) alla posizione di inizio e di fine all'interno del contenuto principale (vedi sez. 2.3.1.5).

In tutte queste tecniche la necessità di rispettare la gerarchia principale induce a ridurre ed offuscare l'importanza delle strutture secondarie, e per questo motivo queste ultime risultano più difficili da identificare e, di conseguenza, anche la gestione e l'elaborazione diventa più complicata: anche banali ricerche che potrebbero essere specificate con un breve XPath spesso richiedono query molto lunghe e complesse, oppure per ottenere semplici visualizzazioni è necessario ricorrere a lunghi e intricati fogli di stile. In alcune situazioni è anche possibile che la struttura principale vincoli quella secondaria a tal punto da impedire di esprimere alcune caratteristiche delle gerarchie secondarie. Per questo la scelta di risolvere il problema dell'overlap con queste tecniche rende il documento più difficile da interpretare correttamente e meno intelleggibile, impedendo di sfruttare le potenzialità degli strumenti forniti da XML e dagli altri principali linguaggi di markup embedded.

TEI: Text Encoding and Interchange

La Text Encoding Initiative (TEI) è stata istituita nel 1987 con il fine di sviluppare, mantenere e promuovere metodologie hardware e software independent di codifica di testi umanistici in formato elettronico. Attraverso le cosiddette Guidelines for Electronic Text Encoding and Interchange, TEI definisce un linguaggio di markup XML per la digitalizzazione dei testi, utile in particolar modo per coloro che intendono costituire archivi e banche dati testuali. Attualmente TEI è utilizzato principalmente nella digitalizzazione di testi letterari, in particolare testi antichi, per poterli conservare nel tempo in maniera efficiente.

TEI affornta il problema dell'overlap a livello di linguaggio di markup, definendo sia gli elementi specifici da utilizzare per descrivere queste situazioni, sia i meccanismi per interpretare correttamente le strutture espresse attraverso queste tecniche di codifica. Di seguito viene presentata l'implementazione TEI delle cinque principali tecniche di codifica presentate precedentemente, mostrando per ognuna una possibile codifica dell'esempio 1.1 realizzata utilizzando il vocabolario definito da TEI P5 [SB05].

2.3.1.1 TEI milestone

```
<TEI>
   ...
   <l>
     <sp>
5      <speaker>Tibère</speaker>
       Poursuivez ...
     </sp>
     <sp>
       <speaker>Agrippine</speaker>
10     Quoi, Seigneur?
     </sp>
     <sp clix:role="start-range" clix:sID="sp1" />
       <speaker>Tibère</speaker>
       Le propos détestable
15 </l>
   <l>
       où je vous ai surprise.
     <sp clix:role="end-range" clix:eID="sp1" />
     <sp clix:role="start-range" clix:sID="sp2" />
20     <speaker>Agrippine</speaker>
       Ah! Ce propos damnable
   </l>
   <l>
       d'une si grande horreur tous mes sens travailla
```

```
25   <sp clix:role="end-range" clix:eID="sp2" />
   </l>
   ...
</TEI>
```

Esempio 2.3. Una codifica del frammento di esempio 2.1 che utilizza TEI Milestone.

Le situazioni di overlap sono state risolte inserendo elementi milestone alle righe 12, 18, 19, 25. In questo caso è stato scelto come primario il vocabolario dei versi, ma si sarebbe ottenuta una codifica altrettanto corretta scegliendo quello delle battute. La struttura degli elementi rappresentati da coppie di milestone risulta appiattita nell'albero XML: questo risolve il problema legato alla buona forma del documento, ma introduce la necessità di utilizzare strumenti software che interpretino gli elementi in milestone e ne ricostruiscano la struttura.

Le milestone permettono di esprimere situazioni di overlap e di self-overlap, ma non consentono di codificare virtual elements. Inoltre introduce relazioni di dominanza fra i nodi di gerarchie differenti non presenti nelle intenzioni dell'autore, come ad esempio fra il primo elemento `<l>` e il primo elemento `<sp>`. L'unico modo di disaccoppiare le relazioni di dominanza dal contenimento è di fornire informazioni esterne al software che effettua il parsing del documento.

2.3.1.2 Flat Milestone

```
<clix:clix>
   <TEI clix:role ="start-range" clix:sID="tei01" />
      <teiHeader clix:role ="start-range" clix:sID="h01" />
         ...
      <teiHeader clix:role ="end-range" clix:eID="h01" />
      <text clix:role="start-range" clix:sID="t01" />
         <body clix:role="start-range" clix:sID="b01" />
            <l clix:role="start-range" clix:sID="l01" />
               <sp clix:role="start-range" clix:sID="sp01" />
                  <speaker clix:role="start-range" clix:sID="spk01" />
                     Tibère
                  <speaker clix:role="end-range" clix:eID="spk01" />
                  Poursuivez ...
               <sp clix:role="end-range" clix:eID="sp01" />
               <sp clix:role="start-range" clix:sID="sp02" />
                  <speaker clix:role="start-range" clix:sID="spk02" />
                     Agrippine
                  <speaker clix:role="end-range" clix:eID="spk02" />
                  Quoi, Seigneur?
               <sp clix:role="end-range" clix:sID="sp02" />
               <sp clix:role ="start- range" clix:sID="sp1" />
                  <speaker clix:role ="start-range" clix:sID ="spk03" />
                     Tibère
                  <speaker clix:role ="end-range "clix:eID="spk03" />
```

```
            Le propos détestable
          <l clix:role="end-range" clix:eID="l01" />
          <l clix:role="start-range" clix:sID=" l02" />
            où je vous ai surprise.
          <sp clix:role="end-range" clix:eID="sp1" />
          <sp clix:role="start-range" clix:sID="sp2" />
            <speaker clix:role="start-range" clix:sID="spk04" />
              Agrippine
            <speaker clix:role="end-range" clix:eID="spk04" />
            Ah! Ce propos damnable
          <l clix:role="end-range" clix:eID="l02" />
          <l clix:role="start-range" clix:sID="l03" />
            d'une si grande horreur tous mes sens travailla
          <sp clix:role="end-range" clix:eID="sp2 " />
          <l clix:role="end-range" clix:eID ="l03" />
        <body clix:role="end-range" clix:eID="b01" />
      <text clix:role="end-range" clix:eID="t01" />
    <TEI clix:role="end-range" clix:eID="tei01" />
</clix:clix>
```

Esempio 2.4. Una codifica con Flat Milestone (CLIX) del frammento di esempio 2.1.

L'esempio precedente utilizza l'approccio CLIX [DER04], che prevede di esprimere tutti gli elementi via milestone. Il documento risultante presenta una struttura piatta: escluso l'elemento radice, il documento è una sequenza di caratteri separata da elementi vuoti. L'unico modo per ricostruire la struttura del documento originale è di utilizzare un software specifico.

Questa tecnica permette di rappresentare self-overlap attraverso uno schema di co-indexing, ma non supporta virtual elements. Come nel caso delle milestone anche questo schema introduce relazioni di dominanza indesiderate che possono essere rimosse solo fornendo istruzioni specifiche al software che effettua il parsing.

2.3.1.3 TEI fragmentation

```
<TEI>
  ...
  <l>
    <sp>
      <speaker>Tibère</speaker>
      Poursuivez...
    </sp>
    <sp>
      <speaker>Agrippine</speaker>
      Quoi, Seigneur?
    </sp>
    <sp xml:id="sp1.1" next="sp1.2">
      <speaker>Tibère </speaker>
      Le propos détestable
    </sp>
  </l>
  <l>
```

```
      <sp xml:id="sp1.2">
        où je vous ai surprise .
      </sp>
      <sp xml:id="sp2.1" next="sp2.2">
        <speaker>Agrippine</speaker>
        Ah! Ce propos damnable
      </sp>
    </l>
    <l>
      <sp xml:id="sp2.2">
        d'une si grande horreur tous mes sens travailla
      </sp>
    </l>
    ...
  </TEI>
```

Esempio 2.5. Una codifica dell'esempio 2.1 che utilizza TEI fragmentation

La fragmentation permette di rappresentare la maggior parte delle caratteristiche complesse di documenti presentate in precedenza: è possibile esprimere situazioni di overlap, self-overlap, discontinuous elements e virtual elements. Come visto nel caso della milestone, anche in questo caso l'uso di un singolo documento XML per rappresentare più di una gerarchia può portare a creare relazioni di dominanza fra elementi appartenenti a gerarchie separate che non erano nelle intenzioni dell'autore.

2.3.1.4 Twin documents

```
<TEI><!-- vocabolario dei versi -->
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <l>
        <speaker>Tibère</speaker>
        Poursuivez...
        <speaker>Agrippine</speaker>
        Quoi, Seigneur?
        <speaker>Tibère</speaker>
        Le propos détestable
      </l>
      <l>
        où 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><!-- vocabolario delle battute -->
   <teiHeader>...</teiHeader>
   <text>
      <body>
         <sp>
             <speaker>Tibère</speaker>
             Poursuivez...
         </sp>
         <sp>
             <speaker>Agrippine</speaker>
             Quoi, Seigneur?
         </sp>
         <sp>
             <speaker>Tibère</speaker>
             Le propos détestable
             où 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>
```

Esempio 2.6. Il frammento di esempio è stato rappresentato utilizzando due twin documents, uno per il vocabolario dei versi e uno per il vocabolario delle battute

La tecnica dei twin documents prescrive di rappresentare ogni gerarchia con un documento XML distinto. Questa tecnica non consente di codificare virtual elements, e nel caso in cui i vocabolari utilizzati dai diversi documenti siano disgiunti non permette di rappresentare situazioni di overlap. A differenza dei due casi precedenti i twin documents non introducono relazioni di dominanza indesiderate fra gli elementi delle diverse gerarchie.

2.3.1.5 TEI stand-off

```
<root >
   <speaker xml:id="spk01">Tibère</speaker>
   Poursuivez...
   <speaker xml:id="spk02">Agrippine</speaker>
   Quoi, Seigneur?
   <speaker xml:id="spk03">Tibère</speaker>
   Le propos détestable
   où je vous ai surprise.
   <speaker xml:id="spk04">Agrippine</speaker>
   Ah! Ce propos damnable
   d'une si grande horreur tous mes sens travailla
</root>
```

Esempio 2.7. un frammento che rappresenta il contenuto cui fanno riferimento le annotazioni in stand-off.

Dato il documento precedente è possibile rappresentare l'esempio con due frammenti in stand-off

```
<TEI>  <!-- vocabolario dei versi -->
   ...
   <l>
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk01'))"/>
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk01')),0,13)" />
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk02'))"/>
     <xinclude href="source.xml"
         xpointer ="string-range(xpath1(id('spk02')),0,15)"/>
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk03'))" />
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk03')),0,19)" />
   </l>
   <l>
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk03')),20,23)"/>
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk04'))" />
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk04')),0,23)" />
   </l>
   <l>
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk04')),24,47)"/>
   </l>
   ...
</TEI>
```

```
<TEI>  <!-- vocabolario delle battute -->
   ...
   <sp>
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk01'))" />
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk03')),0,13)" />
   </sp>
   <sp>
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk02'))" />
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk02')),0,15)"/>
   </sp>
   <sp>
     <xinclude href="source.xml"
         xpointer="xpath1(xpath1(id('spk03'))" />
     <xinclude href="source.xml"
         xpointer="string-range(xpath1(id('spk03'))),0,42)"/>
   </sp>
   <sp>
     <xinclude href="source.xml"
         xpointer="xpath1(id('spk04'))" />
```

```
      <xinclude href="source.xml"
          xpointer="string-range(xpath1(id('spk04')),0,70)"/>
    </sp>
    ...
</TEI>
```

Esempio 2.8. Il frammento dell' esempio2.1 è stato rappresentato utilizzando due frammenti in stand-off che fanno riferimento allo stesso contenuto condiviso.

L'esempio precedente utilizza il meccanismo degli XPointer [DDG02] per definire riferimenti fra i frammenti di documento in stand-off e il contenuto condiviso, come prescritto da TEI P5. Se i vocabolari in stand-off sono disgiunti anche questa tecnica non consente di codificare situazioni di self-overlap. Analogamente alla tecnica dei twin documents l'uso di documenti esterni evita di creare relazioni di dominanza indesiderate fra i nodi delle diverse gerarchie.

2.3.1.6 RDFa

Un altro metodo utilizzato per affrontare il problema dell'overlap è RDFa [AB08]: questa soluzione non è propriamente un approccio all'overlap a livello di linguaggio di markup, in quanto propone di usare un altro modello (RDF) per iniettare annotazioni semantiche all'interno di un documenti XML utilizzando le strutture comuni rese disponibili dal linguaggio stesso. RDFa viene però presentato in questa sezione in quanto può venire utilizzato per affrontare il problema dell'overlap (ad esempio aggiungendo informazioni strutturali al documento), e perché condivide con le altre tecniche di codifica discusse in questa sezione i limiti che derivano dalle strutture dei linguaggi di markup basati sul modello ad albero.

RDFa descrive un meccanismo per inserire statement RDF in un documento HTML utilizzando alcuni attributi di HTML stesso (la recommendation W3C propone di utilizzare *property*, *about*, *typeof*). In questo modo i documenti che utilizzano RDFa risultano arricchiti di nuove asserzioni semantiche machine-readable, ma presentano alcuni limiti: per prima cosa non consentono di esprimere fatti che riguardano strutture che non sono identificate da markup XML, per cui in questi casi l'unica soluzione è quella di aggiungere elementi strutturali non presenti nel documento originale (per esempio elementi *span*); inoltre RDFa ha il limite di non permettere di esprimere asserzioni multiple sullo stesso elemento di markup. L'applicazione di RDFa al problema dell'overlap produce quindi risultati che in molti

casi risultano eccessivamente complicati ed ineleganti, complicando la struttura del documento in maniera simile ai meccanismi presentati precedentemente.

2.3.2 Soluzioni a livello di linguaggio di meta-markup

Un'altra possibilità è quella di affrontare il problema dell'overlapping markup non più a livello della grammatica dei linguaggi di markup, ma di tentare di intervenire a livello di meta-linguaggio di markup: l'idea è quella che sia il meta-linguaggio stesso a dover fornire i meccanismi di supporto alla gestione di gerarchie di markup multiple. In questo modo ogni linguaggio definito con queste sintassi può esprimere in maniera esplicita le situazioni di overlap che si presentano all'interno del documento, senza dovere ricorrere ad ulteriori meccanismi o strumenti che interpretino il significato delle annotazioni.

Di seguito vengono presentate due delle soluzioni proposte: TexMECS [HS01] propone di risolvere alcune problematiche legate all'overlap estendendo linguaggi di markup della famiglia SGML già esistenti, mentre XCONCUR [SW06], in maniera simile a LMNL [TP02], introduce all'interno del documento la possibilità di organizzare il contenuto dei documenti in diversi layer. Per ognuno di questi casi viene mostrata una possibile codifica dell’esempio 2.1.

2.3.2.1 TexMECS

TexMECS è un meta-linguaggio di markup progettato per la codifica di documenti di testo complessi. Come in XML è definito il concetto di elemento, che viene delimitato da tag di inizio e tag di fine. La potenza espressiva di TexMECS è però superiore a quella di XML: oltre a permettere di rappresentare strutture ad albero, TexMECS consente di rappresentare strutture a grafo in cui non è richiesto che i tag di inizio e di fine si annidino in maniera appropriata. Infatti TexMECS dichiara esplicitamente di fare riferimento a GODDAG (vedi sez. 2.3.3) come struttura dati appropriata per rappresentare documenti TexMECS.

Per documenti che presentano una struttura gerarchica ad albero XML e TexMECS sono isomorfi, mentre quando si intende gestire documenti che presentano overlap TexMECS aggiunge alcune specifiche caratteristiche al linguaggio che

permettono di supportare:

- *self-overlap:* utilizzando un semplice schema di co-indexing è possibile rappresentare situazioni in cui due elementi con lo stesso general identifier si sovrappongono. Per collegare le coppie di tag corrispondenti si utilizza il simbolo tilde (“~”) e un suffisso costituito da numeri e lettere.
- *virtual elements:* è possibile fare puntare un elemento ad un altro tramite un meccanismo di linking e delimitatori specifici: gli elementi che fanno riferimento ad altri elementi ne ereditano il contenuto. Attraverso i virtual elements è possibile definire due elementi che condividono lo stesso contenuto senza creare fra di essi relazioni di dominanza. Per fare riferimento ad un altro elemento è necessario delimitare il general identifier da due simboli “^” e giustapporre l’id del riferimento.
- *discontinuous elements:* questi elementi possono venire rappresentati utilizzando elementi interrotti, che vengono indicati da specifici tag di inizio, di sospensione, di ripresa e di fine.
- *elementi non ordinati:* sono elementi per cui l'ordine del contenuto è insignificante, per cui può essere riordinato durante la serializzazione senza nessuna perdita di informazione. Questi elementi non possono venire interrotti.

Attraverso questa semplice notazione TexMECS, oltre a garantire allo stesso modo di XML che sia possibile interpretare in maniera corretta il documento, assicura un mapping uno-a-uno fra le caratteristiche all'interno del testo e le coppie di tag di inizio e di chiusura nella codifica.

```
<TEI|
  <teiHeader|...|teiHeader>
  <text|
    <body|
      <l|
        <sp|<speaker|Tibère|speaker>Poursuivez...|sp>
        <sp|<speaker|Agrippine|speaker>Quoi, Seigneur?|sp>
        <sp|<speaker|Tibère|speaker>Le propos détestable|-sp>
      |l>
      <l|
        <+sp|où je vous ai surprise.|sp>
        <sp|<speaker|Agrippine|speaker>Ah! Ce propos damnable |-sp>
      |l>
      <l|
        <+sp|d’une si grande horreur tous mes sens travailla|sp>
      |l>
    |body>
  |text>
|TEI>
```

Esempio 2.9. una codifica TexMECS dell’esempio 2.1.

In TexMECS quando due elementi sono annidati correttamente viene inferita una relazione di dominanza fra di essi, esattamente come avviene in XML. La codifica precedente introduce quindi relazioni di dominanza indesiderate fra il primo elemento <l> e i primi due elementi <sp>. L'unico modo di evitare che questo si verifichi è di rappresentare gli elementi di una delle due gerarchie con dei virtual elements.

2.3.2.2. XConcur

Un altro approccio è quello utilizzato da *XConcur* [SW06], che propone di utilizzare una caratteristica opzionale di SGML chiamata CONCUR che permette di esprimere "viste multiple e concorrenti dalla struttura di un documento" [BBG95] [BHK88]. Un documento XConcur è composto da una serie di livelli, chiamati "annotation layer", che coesistono nella stessa struttura: per assegnare un elemento a un livello è sufficiente fare precedere l'identificativo del livello al general identifier dell'elemento. In maniera simile a XML, XConcur richiede la buona forma della struttura di ogni annotation layer: ogni livello rappresenta una gerarchia indipendente che può venire estratta come una unità singola ed essere validata secondo un DTD, uno schema RelaxNG o XML-Schema [Sw06]. La principale differenza rispetto a SGML è che in XConcur non sono consentiti elementi senza un annotation layer id specifico, per cui il data-model dei documenti è il multi-rooted tree.

Per esaminare la buona forma di un livello è sufficiente eliminare tutti i tag che non appartengono a tale annotation layer, e rimuovere dagli elementi rimasti i prefissi che fanno riferimento all'annotation layer, e infine applicare lo stesso metodo a tutte le dichiarazioni di DOCTYPE: la struttura risultante deve rispettare i criteri di buona forma di XML, e nel caso sia specificato uno schema per quel livello è possibile effettuare una validazione. Le relazioni e i vincoli fra le gerarchie multiple in cui è strutturato un documento XConcur sono regolate da un linguaggio di vincoli collegato ad XConcur chiamato XConcur-CL. I documenti XConcur risultano essere molto complessi: per questo motivo sono stati realizzati numerosi strumenti che agevolano la manipolazione di tali documenti.

```
<?XCONCUR version="1.1" encoding="UTF-8"?>
<?XCONCUR-schema layer-id="l1" root="verse" system="teiverse.dtd"?>
<?XCONCUR-schema layer-id="l2" root="performance"
system="teiperformance.dtd"?>
```

```
<(l1)verse>
 <(l2)performance>
      <(l1)l>
      <sp(l2)><(l1)speaker>Tibère</(l1)speaker>Poursuivez...</(l2)sp>
      <sp(l2)><(l1)speaker>Agrippine<(l1)speaker>Quoi, Seigneur?</(l2)sp>
      <sp(l2)><(l1)speaker>Tibère</(l1)speaker>Le propos détestable
   </(l1)l>
   <(l1)l>
      où je vous ai surprise.</(l2)sp>
      <sp(l2)><(l1)speaker>Agrippine</(l1)speaker>Ah! Ce propos damnable
   </(l1)l>
   <(l1)l>
        d'une si grande horreur tous mes sens travailla</(l2)sp>
   </(l1)l>
 </(l2)performance>
```

Esempio 2.10. Una codifica XConcur del frammento dell'esempio 2.1 .

Nell'esempio precedente le due gerarchie dei versi e dei dialoghi sono state codificate in due layer differenti e indipendenti.

XConcur non viene considerato come una soluzione apprezzabile del problema dell'overlap: le principali riserve, descritte in [Mur95] in relazione all'opzione CONCURR di SGML e valide anche per XConcur, contestano il fatto che questa soluzione non consenta di modellare in maniera appropriata alcune delle situazioni reali legate al problema dell'overlap più comuni come cancellazioni, inserimenti, duplicazioni e riordinamento di contenuto all'interno delle varie viste. Ciò è dovuto al fatto che questo modello impone che tutti gli annotation layer siano vincolati a rispettare il document order imposto dal contenuto testuale: l'ordine in cui il contenuto si presenta all'interno dei diversi livelli deve necessariamente essere identico all'ordine del contenuto testuale, per cui le situazioni che prevedono il riordinamento di materiale presente in altri posizioni all'interno del documento (per esempio virtual e discontinuous elements) possono essere espresse solamente attraverso ulteriori meccanismi (ad esempio con alcune tecniche descritte precedentemente, come la fragmentation o lo stand-off markup).

2.3.3 Soluzioni a livello di modello astratto: GODDAG

Gli approcci presentati in precedenza, pur operando a livelli diversi, affrontano tutti il problema dell'overlap con annotazioni di tipo embedded e condividono l'idea di forzare i documenti da rappresentare in strutture ad albero. Gli autori di queste

proposte concordano sul fatto che il modello ad albero sia una semplificazione limitante, ma le soluzioni messe in campo sembrano fermarsi un passo prima di risolvere il problema: ognuna presenta infatti qualche limite, e nessuna rappresenta una risposta completa al problema dell'overlap nella sua interezza.

Un'altra filosofia è invece quella di abbandonare il modello ad albero a favore di un nuovo schema astratto più generale. La proposta più rilevante di questo approccio è *GODDAG* (General Ordered-Descendant Direct Acyclic Graph) che utilizza come struttura dati un grafo diretto aciclico. In [SH00] Sperberg-McQueen e Huitfeldt spiegano che l'overlap può venire rappresentato da grafi che sono molto simili ad alberi, ma in cui alcuni nodi hanno più di un padre, per cui la scelta di una struttura dati a grafo consente di esprimere quella che per gli autori è l'essenza dell'overlap, riassumibile nell'assioma “overlap is multiple parentage”.

In maniera simile alle rappresentazioni convenzionali di SGML ed XML che utilizzano un modello ad albero, in GODDAG gli elementi sono rappresentati dai nodi di un grafo, il contenuto testuale del documento da etichette sulle foglie del grafo e le relazioni di contenimento vengono rappresentate da archi. Come detto in precedenza, a differenza di SGML e XML questa struttura consente ai nodi figlio di avere più di un padre, e inoltre è imposto un ordinamento fra i figli di ogni nodo (da questa caratteristica deriva appunto il termine Ordered-Descendant). E' stato dimostrato che non è sempre possibile linearizzare questi grafi, per cui è stata proposta una versione ristretta di GODDAG, chiamata *r-GODDAG*, che garantisce di poter essere linearizzata in TexMECS (vedi sezione 2.3.2.1). Per chiarezza riportiamo di seguito la definizione del modello di r-GODDAG.

Definizione 2.1 (restricted GODDAG). Un GODDAG è ristretto se rispetta le seguenti condizioni:

1. i nodi foglia formano una sequenza (chiamata *frontiera*) e sono totalmente ordinati secondo la relazione $\prec$ (document order); $\prec$ è una tricotomia sui nodi foglia:

   $\forall$ *leaf1* , *leaf2* nodi foglia vale o $leaf1 \prec leaf2$, o $leaf2 \prec leaf1$, oppure
   $leaf1 = leaf2$;

2. ogni nodo *n* domina una sottosequenza contigua della frontiera (chiamata *leafset* di n):

   se *n* domina i nodi foglia *leaf1*, *leaf2,* allora ∀ *leaf* nodo foglia tale per cui *leaf1* ≺ *leaf* ≺ *leaf2*, *n* domina *leaf*.

3. nessuna coppia di nodi domina la stessa sottosequenza della frontiera:

   ∀ *n1*, *n2* nodi, esiste un nodo foglia *leaf* tale che o *n1* domina *leaf* e *n2* non domina *leaf*, oppure viceversa.

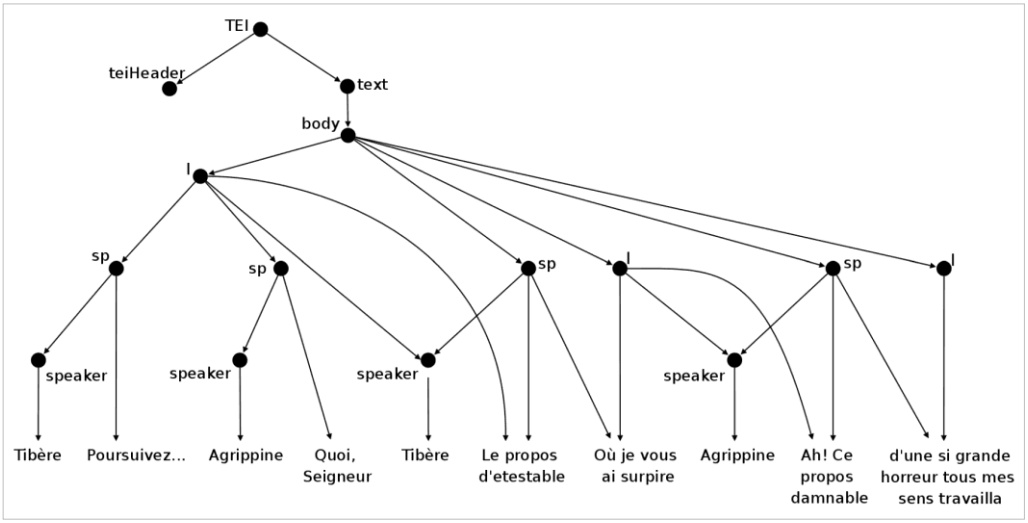

Figura 2.1: Un r-GODDAG che corrisponde all'esempio 2.1. I nodi foglia vuoti che garantiscono che si tratti di un r-GODDAG non sono stati mostrati per fornire una rappresentazione concisa.

La terza condizione impedisce che due elementi dominino lo stesso insieme di foglie, ma come suggerito dagli autori di r-GODDAG può venire aggirata aggiungendo opportunamente nodi foglia vuoti. La seconda condizione introduce invece un limite rilevante in quanto impedisce di rappresentare discontinuous elements.

La figura 2.1 mostra una possibile rappresentazione r-GODDAG dell'esempio 2.1. In questo esempio vengono rappresentate situazioni di overlap in maniera esplicita: il terzo nodo "speaker" ha infatti due genitori: il nodo "i" e il nodo "l". Anche in questo caso sono state però introdotte relazioni di dominanza che non erano nelle intenzioni dell'autore.

2.3.4 Discussione e valutazione di questi approcci

Nelle sezioni precedenti abbiamo descritto i tre principali approcci al problema dell'overlapping markup e abbiamo confrontato le varie soluzioni analizzando per ognuna diverse codifiche dello stesso frammento (esempio 2.1). Di seguito riassumeremo i principali limiti delle diverse soluzioni.

Le principali limitazioni dei primi due approcci sono dovuti al fatto che affrontano il problema dell'overlap come una mera questione tecnica legata alle caratteristiche dei diversi formati, per cui queste soluzioni propongono compromessi che migliorano ma non risolvono completamente i problemi legati alla gestione di gerarchie multiple che sono connaturati ai meta-linguaggi di markup di tipo embedded.

Il problema è dovuto alla complessità intrinseca delle strutture che rappresentano gerarchie multiple, e risulta evidente come soluzioni basate su strutture ad albero o da esse derivate fatichino a gestire in maniera naturale ed efficace la complessità di questi modelli. A sostegno di queste considerazioni è possibile osservare che nessuna di queste proposte consente di esprimere tutte le caratteristiche di documenti complessi che si intende descrivere (simple overlap, self-overlap, virtual element, discontinuous elements, ecc.), ma rappresenta una soluzione parziale a tali questioni. Inoltre spesso si evidenzia anche un aumento di complessità, sia a livello di linguaggio che nel caso si tratti di meta-linguaggio, per cui anche la gestione di documenti non troppo complessi richiede il supporto di strumenti e software specifici.

GODDAG si dimostra invece come un modello appropriato per la gestione della complessità di strutture in overlap, ma purtroppo sul piano pratico non è supportato da strumenti e tecnologie che ne agevolino l'utilizzo nei diversi contesti applicativi. Inoltre GODDAG non consente linearizzazioni in formati embedded in generale, ma solo una versione ristretta del modello (r-GODDAG) può venire linearizzata (in TexMECS).

2.4 Transformation algorithm

Un altro approccio interessante al problema dell'overlapping markup è quello proposto in [MVZ08]: in questo articolo viene presentato un framework di conversione sintattica fra formati di markup embedded che gestiscono il fenomeno dell'overlapping. Questa soluzione permette di gestire sia le tecniche di codifica dell'overlap utilizzate dai formati XML come milestone, flat milestone, fragmentation, stand-off markup e twin documents (vedi sezione 2.3.1), sia formati non XML come TexMECS e LMNL (vedi sezione 2.3.2). Il framework suggerito ha una struttura a stella in cui ai vertici troviamo i formati di destinazione considerati. Invece che descrivere (#formati * (#formati – 1)) algoritmi di conversione, gli autori pongono al centro di questo modello r-GODDAG: tutti i formati vengono prima convertiti nel formato centrale, e poi in un secondo passo vengono serializzati nel formato di destinazione. Questa scelta ha due motivazioni principali: per prima cosa il numero degli algoritmi da implementare si riduce a (#formati + #formati-1), semplificando quindi lo sforzo complessivo; la seconda ragione è che r-GODDAG ha una struttura a grafo che consente di gestire in maniera diretta e immediata le situazioni di overlap. Questo semplifica la soluzione del problema in quanto non ci sono tecniche di codifica che offuscano la gerarchia principale a scapito delle altre, ma tutte le feature sono esprimibili con semplicità ed in maniera esplicita. In questo modo non è necessario prendere in considerazione i limiti espressivi e pratici legati ad ogni formato, per cui le problematicità del processo di conversione vengono isolate. Ad esempio, ipotizzando di volere convertire un documento codificato con la tecnica fragmentation in un documento equivalente che utilizza milestone, il processo di conversione può essere realizzato in due passi:

1. da fragmentation a r-GODDAG, in cui la complessità è tutta legata al formato fragmentation.
2. da r-GODDAG a milestone, in cui la complessità è legata esclusivamente al formato milestone.

E' interessante l'idea di individuare un formato principale che abbia espressività uguale o superiore ai formati da convertire cui ricondursi, in quanto permette di isolare la complessità legate ai formati ai margini del framework di conversione: una

volta che si è giunti in r-GODDAG è possibile analizzare l'overlap per quello che è in realtà, ossia strutture a grafo, senza preoccuparsi delle complessità legate alle rappresentazioni che derivano da tecniche di codifica o sintassi particolari. In questo modo le operazioni utilizzate durante la conversione, come la ricerca e la manipolazione del documento, possono essere realizzate con un modello che rappresenta in maniera diretta la struttura del documento.

Capitolo 3

FRETTA:

UN MOTORE PER LA LINEARIZZAZIONE DI DOCUMENTI EARMARK

Nel capitolo precedente abbiamo introdotto il problema dell'overlapping markup e abbiamo discusso il principale limite che dimostrano i linguaggi di markup embedded nel gestire i casi in cui più gerarchie si sovrappongono sullo stesso contenuto documentale. Il problema principale è dovuto al fatto che la filosofia di organizzare i documenti in strutture ad albero come in XML si dimostra una semplificazione troppo limitante che non è in grado di gestire queste situazioni.

Per risolvere questo problema è stato sviluppato EARMARK [DPV11a][DPV11b], un meta-linguaggio di markup che permette di esprimere strutture più complesse di alberi, ovvero digrafi, basato sulle tecnologie del Semantic Web. Questo progetto prevede di sviluppare un insieme di tecnologie e di strumenti simili a quelli disponibili per XML che forniscano supporto allo sviluppo di applicazioni basate su EARMARK. In particolare è stato previsto lo sviluppo di un framework che consenta di mettere in rapporto EARMARK con i principali linguaggi di markup di tipo embedded. L'obiettivo è quello di realizzare uno strumento che permetta di convertire documenti EARMARK in documenti XML, e viceversa. Il processo di trasformazione realizzato dal framework EARMARK è diviso in due fasi: ROCCO (Read, Overhaul, Convert, Classify, Organize), che si occupa di eseguire la trasformazione di file XML in documenti EARMARK, e FRETTA (From EARMARK To Tag), che esegue l'operazione inversa, convertendo documenti EARMARK nel formato XML. Questa tesi propone un'implementazione di FRETTA, un motore che realizza la seconda fase di questa conversione.

In questo capitolo introdurremo il modello di EARMARK, analizzando in particolare i vantaggi che questo approccio propone nella gestione delle situazioni di overlap (sezione 3.1). In seguito presenteremo brevemente la struttura del framework EARMARK (sezione 3.2), e concluderemo questa discussione descrivendo l'architettura di FRETTA, l'applicazione sviluppata in questo lavoro di tesi (sezione 3.3).

3.1 EARMARK

EARMARK (Extremely Annotational RDF MARKup) [DPV11a] [DPV11b] è un meta-linguaggio di markup basato sulle tecnologie del Semantic Web che introduce un nuovo approccio sintattico all'overlapping markup che combina i vantaggi delle annotazioni embedded e di quelle esterne in un unico modello. Un aspetto fondamentale di questa proposta è la fiducia nelle tecnologie legate al Semantic Web, che forniscono un framework completo che permette di realizzare soluzioni integrabili ai moduli esistenti e facilmente estendibili. EARMARK è un meta-linguaggio intermedio costruito sul data model di RDF [MM04] e OWL

[HKP09], per cui i documenti EARMARK godono di una completa integrazione con questi standard e con gli strumenti ad essi legati.

Oltre fornire un modello che permette di gestire le caratteristiche di documenti basati su alberi come XML e altri meta-linguaggi di markup gerarchici, EARMARK consente di esprimere in maniera esplicita strutture in overlap senza la necessità di utilizzare specifiche tecniche di codifica: durante lo sviluppo di EARMARK gli autori si sono infatti concentrati sulle modalità da utilizzare per memorizzare le strutture in overlap. L'obiettivo principale è stato quello di trovare una soluzione che permettesse di esprimere le strutture in overlap in maniera esplicita, in modo che risultino trasparenti e facilmente comprensibili sia agli utenti che alle applicazioni. Per raggiungere questo scopo EARMARK abbandona l'approccio embedded seguito dai linguaggi di markup più diffusi, come ad esempio XML e tutti gli altri linguaggi SGML-based, in favore di una soluzione di tipo stand-off: i markup descriptor che costituiscono le strutture da descrivere non contengono direttamente le parti di documento o le sottostrutture cui si riferiscono, ma puntano ad esse attraverso altri meccanismi. L'idea di base è dunque quella di modellare un documento EARMARK come una collezione di frammenti di testo indirizzabili, e di associare a tale contenuto testuale delle asserzioni OWL che descrivano sia caratteristiche strutturali che proprietà semantiche legate a quel contenuto. In questo modo EARMARK è in grado di descrivere, oltre a casi semplici di documenti con una singola gerarchia di markup, anche situazioni di gerarchie multiple in cui il contenuto testuale legato agli elementi del markup appartiene ad alcune gerarchie ma non ad altre. In questo modo tutte le caratteristiche complesse legate al problema dell'overlap descritte in precedenza (simple overlap, self overlap, virtual element, discontinuous element, ecc.) possono essere espresse esplicitamente in EARMARK. Allo stesso modo è possibile aggiungere in qualsiasi momento annotazioni semantiche attraverso ulteriori asserzioni, e anche queste possono essere in overlap con il contenuto precedente.

Un vantaggio evidente di EARMARK è la possibilità di accedere e navigare i documenti utilizzando strumenti molto diffusi e ampiamente supportati legati al Semantic Web. I documenti EARMARK sono infatti modellati attraverso ontologie OWL, e le annotazioni che costituiscono i documenti sono semplici asserzioni RDF. Di conseguenza è possibile utilizzare linguaggi di query come SPARQL [PS08] e un

vasto insieme di strumenti esistenti come Jena[3] e Pellet[4] per gestire strutture in overlap anche molto complicate: operazioni di ricerca e manipolazione di strutture arbitrariamente complesse che sono molto difficili o addirittura impossibili con le tecnologie per XML risultano semplici ed immediate in EARMARK.

Nella parte restante di questa sezione viene fornita una breve descrizione del modello concettuale di EARMARK, e vengono presentate le EARMARK API, una libreria Java che permette di creare e manipolare documenti EARMARK all'interno di applicazioni Java, mentre nell'ultima parte viene discusso come questa soluzione fornisca un approccio migliore al problema dell'overlap rispetto i linguaggi di markup tradizionali.

3.1.1 Un approccio ontologico ai linguaggi di meta-markup

La struttura di EARMARK è definita attraverso un'ontologia OWL[5] che specifica le classi e le relazioni che costituiscono il modello. L'ontologia è suddivisa in due parti: una parte descrive le ghost classes, che definiscono la struttura generale dei documenti EARMARK, l'altra le shell classes, ossia le classi che vengono effettivamente istanziate per creare documenti EARMARK.

3.1.1.1 Ghost classes

Le *ghost classes* descrivono tre concetti disgiunti – *docuversi*, *range* e *markup item* – attraverso tre classi OWL.

Il contenuto testuale di un documento EARMARK è concettualmente separato dalle annotazioni associate, e viene descritto dalla classe *Docuverse*. Il nome di questa classe deriva dal concetto introdotto da Ted Nelson nel progetto Xanadu [NEL80] per riferirsi a collezioni di frammenti di testo che possono essere interconnessi e transclusi all'interno di nuovi documenti. Gli individui di questa classe rappresentano gli oggetti del discorso, ossia tutti i contenitori di testo di un documento EARMARK. Gli individui della classe *Docuverse,* che chiameremo semplicemente *docuversi* (con l'iniziale minuscola per distinguerli dalla classe),

[3] http://jena.sourceforge.net/
[4] http://clarkparsia.com/pellet
[5] L'ontologia EARMARK è disponibile all'indirizzo http://www.essepuntato.it/2008/12/earmark

specificano il proprio contenuto con la proprietà *hasContent*.

```
Class: Docuverse
DatatypeProperty: hasContent Characteris
```

La classe *Range* rappresenta il contenuto testuale compreso fra due locazioni all'interno di un docuverso. Un *range*, ossia un individuo della classe *Range*, è definito da una locazione di inizio e una locazione di fine di uno specifico docuverso attraverso le proprietà *begins, ends* e *refersTo* rispettivamente.

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

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

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

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

La classe *MarkupItem* è la superclasse comune a tutti i tipi di annotazione (elementi, attributi e commenti). Un individuo della classe *MarkupItem* (che chiameremo *markupItem*) è una collezione di individui che appartengono alle classi *MarkupItem* e *Range*. Un *markupItem* può essere definito come un insieme, un bag o una lista di altri *markupItem* e *range* usando le proprietà *element* per gli insiemi, *item* e *itemContent* per bag e liste. Nell'ultimo caso la proprietà *nextItem* permette di specificare un ordinamento tra gli elementi delle liste.

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

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

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

Class: c:Collection
Class: c:Set SubClassOf: c:Collection
Class: c:Bag SubClassOf: c:Collection
Class: c:List SubClassOf: c:Bag
```

```
Class: c:Item
Class: c:ListItem SubClassOf: c:Item

ObjectProperty: c:element Domain: c:Collection

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

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

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

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

Questa scelta consente di caratterizzare i *markupItem* con le proprietà definite per le diverse classi: un *markupItem* di tipo *c:Set* rappresenta una collezione di elementi che non contiene ripetizioni e senza nessun ordinamento definito; un *markupItem* di tipo *c:Bag* è una collezione che ammette la ripetizione di elementi ma ancora senza un ordine definito; un *markupItem* di tipo *c:List* è una collezione di elementi ripetibili e ordinati. La definizione di collezioni utilizzata – indicata dal prefisso "c*"* – deriva da un'ontologia importata da EARMARK[6] definita per il progetto SWAN [CKW08].

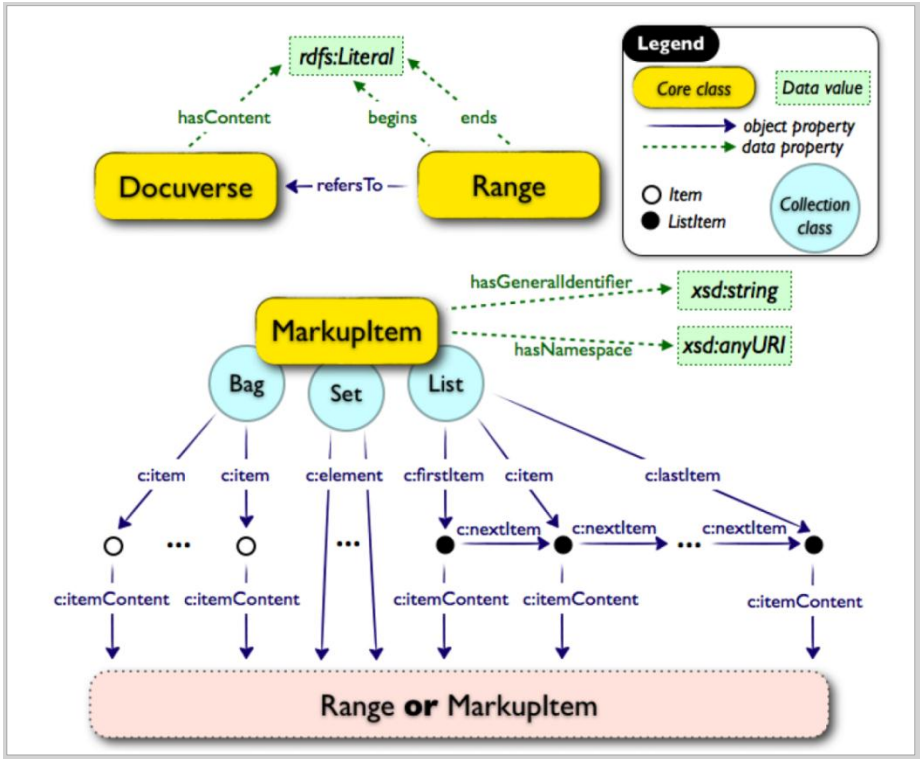

Figura 3.1 Una rappresentazione grafica dell'ontologia di EARMARK

6 L'ontologia che definisce le collezioni utilizzata da EARMARK è disponibile all'indirizzo http://swan.mindinformatics.org/ontologies/1.2/collections.owl

3.1.1.2 Shell classes

Le *shell classes* specializzano le *ghost classes* per fornire informazioni più specifiche sulle istanze EARMARK. La classe *Docuverse* è specializzata nelle classi *StringDocuverse* (il cui contenuto è specificato da una stringa attraverso la proprietà *hasContent*) e *URIDocuverse* (il cui contenuto si trova all'URI specificato attraverso la proprietà *hasContent*). Le sottoclassi della classe *Range* possono gestire diversi tipi di docuversi (testo semplice, XML, LaTEX, ecc.) e vari schemi di indirizzamento: *PointerRange* e *XpathPointerRange*. Infine la classe *MarkupItem* si specializza in tre classi disgiunte: *Element*, *Attribute* e *Comment*.

Per capire come utilizzare EARMARK per descrivere gerarchie di markup introduciamo un frammento XML che utilizza TEI fragmentation per esprimere elementi in overlap sulla stringa “Fabio says that overlhappens”.

```
<phrase>
  Fabio says that
  <noun xml:id="e1" next="e2">overl</noun>
  <verb>h<noun xml:id="e2">ap<noun>pens</verb>
</phrase>
```

Esempio 3.1.

In questo esempio i due elementi `<noun>` rappresentano lo stesso elemento frammentato in due parti e parzialmente sovrapposto con il contenuto testuale di `<verb>` (sui caratteri “ap”). La traduzione EARMARK in Turtle [BT10] è la seguente:

```
@prefix ex:<http://www.example.com /> .
ex:doc :hasContent "Fabio says that overlhappens" .
ex:r0-16 a :PointerRange ; :refersTo ex:doc
  ; :begins "0"^^xsd:integer ; :ends "16"^^xsd:integer .
ex:r16-21 a :PointerRange ; :refersTo ex:doc
  ; :begins "16"^^xsd:integer ; :ends "21"^^xsd:integer .
ex:r21-28 a :PointerRange ; :refersTo ex:doc
  ; :begins "21"^^xsd:integer ; :ends "28"^^xsd:integer .
ex:r22-24 a :PointerRange ; :refersTo ex:doc
  ; :begins "22"^^xsd:integer ; :ends "24"^^xsd:integer .
ex:phrase a :Element ; :hasGeneralIdentifier "phrase"
  ; c:firstItem [ c:itemContent ex:r0-16
  ; c:nextItem [ c:itemContent ex:noun
  ; c:nextItem [ c:itemContent ex:verb ] ] ] .
ex:noun a :Element ; :hasGeneralIdentifier "noun"
  ; c:firstItem [ c:itemContent ex:r16-21
  ; c:nextItem [ c:itemContent ex:r22-24 ] ] .
ex:verb a :Element ; :hasGeneralIdentifier "verb"
  ; c:firstItem [ c:itemContent ex:r21-28 ] .
```

Esempio 3.2.un documento EARMARK che rappresenta l’esempio 3.1.

In figura 3.2 mostriamo una rappresentazione dell'esempio 3.2. Le convenzione grafiche utilizzate in questa figura verranno adottate per rappresentare documenti EARMARK per il resto di questa dissertazione.

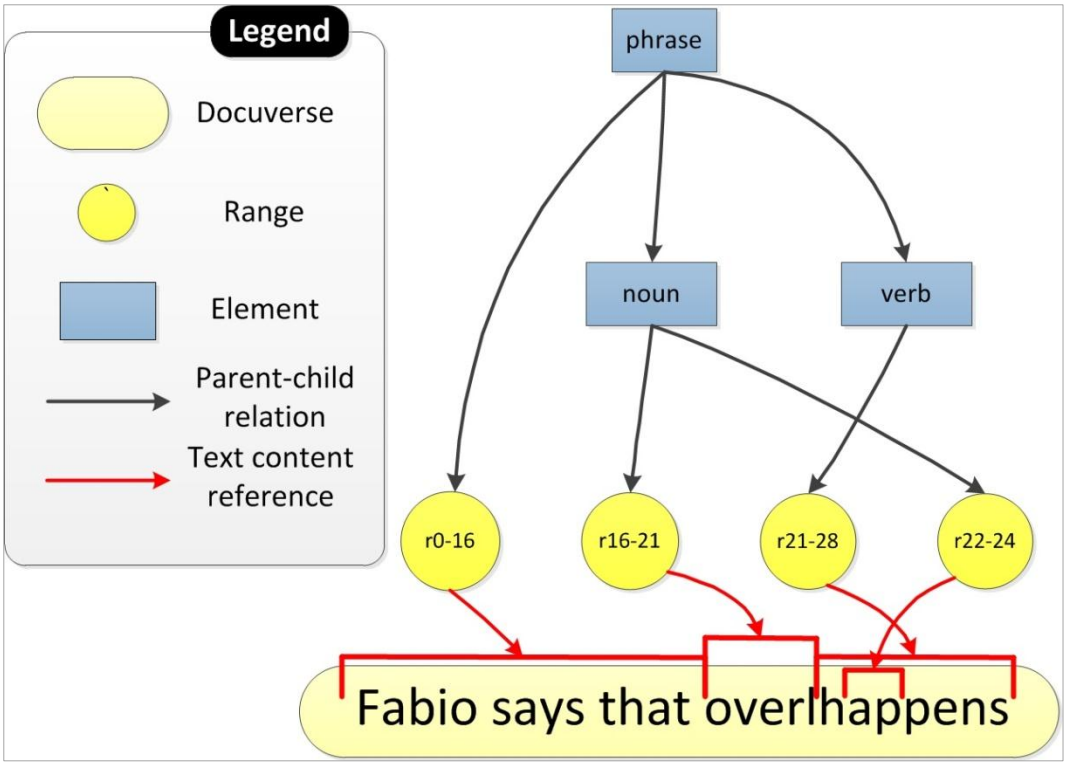

Figura 3.2

3.1.2 EARMARK API

E' già stato progettata e implementata una libreria che permette di creare, validare e manipolare le strutture dati di EARMARK. Le API sviluppate, disponibili presso SourgeForge[7] sotto licenza Apache 2.0, implementano il modello di EARMARK corrente.

Tutto il codice è scritto in Java e utilizza la libreria Jena[8]. L'implementazione della struttura dati ricalca il modello definito nell'ontologia EARMARK, implementando shell e ghost classes, e codificando le proprietà come metodi di tali classi.

Le classi Java EARMARKDocument, Range e MarkupItem sono state

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

sviluppate prendendo in considerazione una particolare interfaccia, chiamata EARMARKNode, che ricalca lo schema dell'interfaccia Node dell'implementazione Java di DOM[9]. Questa scelta permette di mantenere la struttura dati di EARMARK molto simile a uno dei modelli più conosciuti e utilizzati per i documenti XML. Di seguito forniamo un esempio di come sia possibile creare il documento EARMARK presentato nella sezione precedente (esempio 3.2) con le EARMARK API. Per prima cosa creiamo un nuovo documento EARMARK e un docuverso contenente il contenuto testuale del nostro documento:

```
EARMARKDocument ed =
  new EARMARKDocument(URI.create("http://www.example.com"));

String ex = "http://www.example.com/";

Docuverse doc = ed.createStringDocuverse(ex+"doc",
  "Fabio says that overlhappens");
```

Il frammento seguente mostra come creare un range partendo dal docuverso precedente:

```
Range r0_16 =ed.createPointerRange(ex+"r0_16", doc, 0, 16);
```

Infine definiamo i markup item necessari per costruire la struttura del documento. Solitamente per creare un markup item è necessario specificare tre valori: una stringa che rappresenta il general identifier, un identificatore e il tipo della collezione (set, bag o list). Nel nostro caso mostriamo la creazione di tre diversi elementi, che vengono composti in maniera da definire la struttura gerarchica del documento attraverso il metodo appendChild:

```
Element phrase = ed.createElement(
  "phrase", ex+"phrase", Collection.Type.List);
ed.appendChild(phrase); phrase.appendChild(r0_16);
Element noun = ed.createElement(
  "noun", ex+"noun", Collection.Type.List);
phrase.appendChild(noun);
Element verb = ed.createElement(
  "verb", ex+"verb", Collection.Type.List);
phrase.appendChild(verb);
```

Come dimostrano questi esempi, le API di EARMARK permettono di gestire efficacemente le strutture del modello di EARMARK.

[9] http://java.sun.com/xml

3.1.3 Overlap in EARMARK

Il modello di EARMARK permette di rappresentare esplicitamente caratteristiche documentali complesse che derivano da strutture in overlap sullo stesso contenuto documentale. Il frammento presentato nell'*esempio 2.1* ad esempio presenta due strutture indipendenti, la struttura dei versi e la struttura dei dialoghi, che vengono mostrate separatamente in *figura 3.3* e *figura 3.4*.

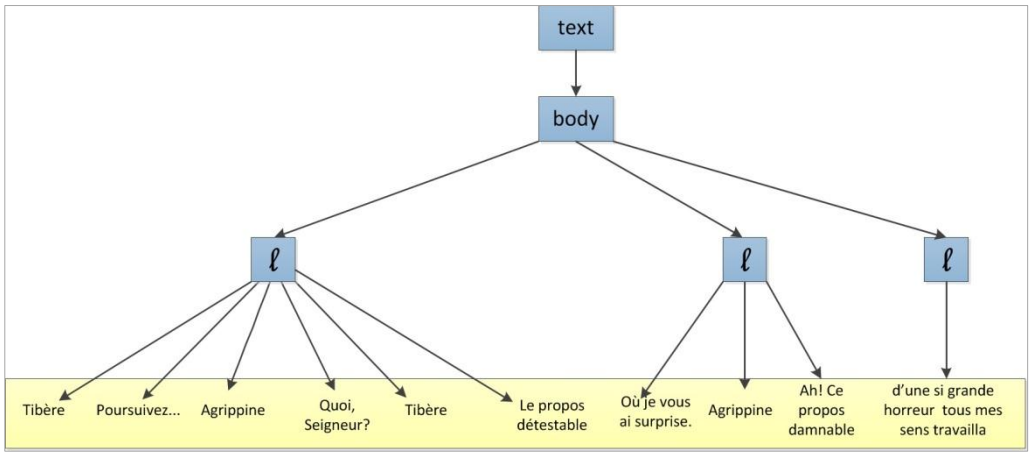

Figura 3.3 Una rappresentazione della struttura dei versi dell'esempio 2.1

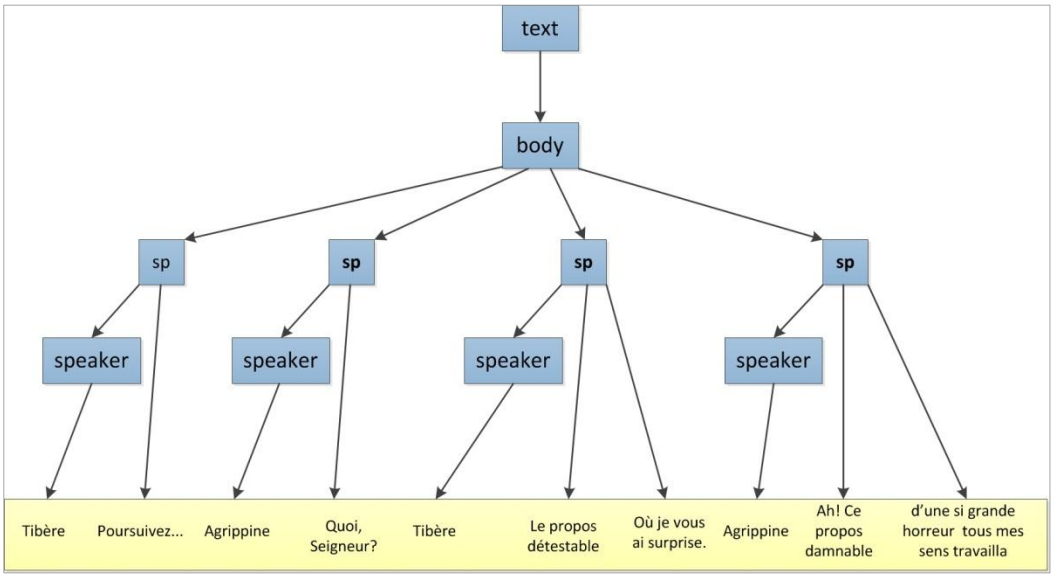

Figura 3.4 Una rappresentazione della struttura dei dialoghi dell'esempio 2.1

Come detto in precedenza, XML non è in grado di esprimere più di una singola gerarchia in un unico documento, per cui l'unico modo di rappresentare una situazione come quella dell'esempio è di utilizzare una delle tecniche di codifica presentate al capitolo 2 (vedi sezione 2.3.1). In EARMARK invece le strutture in overlap possono essere descritte in maniera diretta: la figura 3.5 rappresenta una versione EARMARK dell'esempio precedente.

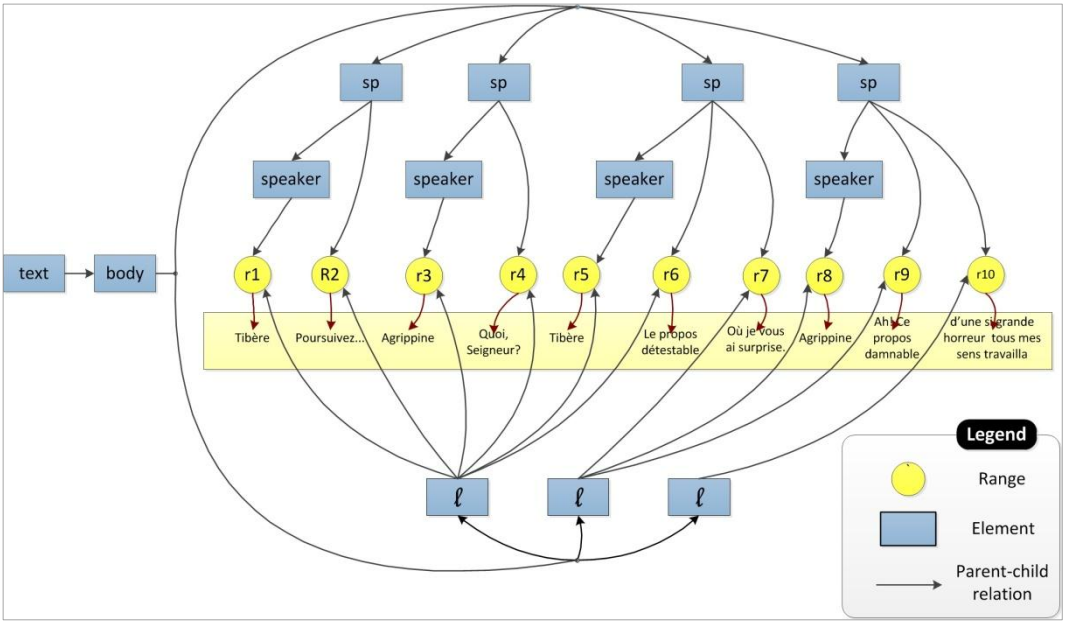

Figura 3.5 Una rappresentazione di una codifica in EARMARK dell'esempio 2.1

Tutti gli elementi (sia quelli condivisi che quelli specifici alle singole gerarchie) vengono rappresentati esplicitamente in un unico documento EARMARK, nonostante alcuni siano in overlap. Il contenuto testuale è rappresentato da dieci range che fanno riferimento al testo contenuto in un unico docuverso.

3.2 Struttura del framework EARMARK

Il progetto di ricerca all'interno del quale si inserisce il lavoro di questa tesi si pone l'obiettivo di sviluppare un insieme di tecnologie e strumenti che supportino la realizzazione di applicazioni basate su EARMARK. In particolare è stato previsto lo sviluppo di un framework di conversione (che chiameremo semplicemente

framework EARMARK) che permetta di convertire documenti EARMARK contenenti strutture in overlap in formati XML, e viceversa. Poiché XML prevede una rigida organizzazione gerarchica ad albero della struttura del documento, l'unico modo di esprimere situazioni di overlap nei formati gestiti dal framework EARMARK è quello di ricorrere a particolari tecniche di codifica (vedi sezione 2.3.1). Tutte queste soluzioni condividono lo stesso approccio di nascondere le informazioni che riguardano le gerarchie secondarie in un'altra forma: elementi vuoti che indicano i limiti di un elemento, elementi divisi in frammenti, riferimenti indiretti, ecc. La necessità di rispettare la gerarchia principale induce a ridurre ed offuscare l'importanza delle strutture secondarie, e per questo motivo le gerarchie espresse attraverso tecniche di codifica risultano più difficili da identificare e, di conseguenza, anche la gestione e l'elaborazione diventa più complicata.

In EARMARK l'essenza dell'overlap, riassunta nella frase “overlap is multiple parentage” [SH00], può venire invece rappresentata in maniera esplicita: poiché il data-model utilizzato è un grafo**,** la scelta di ricondurre i documenti da convertire a questo formato permette di fare emergere le strutture in overlap che nei formati di partenza dovevano venire nascoste nella gerarchia principale.

Il meccanismo di conversione implementato dal framework EARMARK si articola in due fasi: la prima prevede di considerare un file XML contenente caratteristiche complesse, ricostruirne la struttura originale attraverso l'interpretazione del significato delle tecniche di codifica utilizzate per esprimere le gerarchie in overlap e di esprimere tale struttura esplicitamente in EARMARK; la seconda procede in maniera speculare alla prima, effettuando una conversione del documento EARMARK in funzione del formato di destinazione, e linearizzando infine la versione ottenuta nel formato desiderato. Uno dei vantaggi principali di questo approccio è che EARMARK permette di analizzare l'overlap per quello che è in realtà, ossia strutture a grafo, senza preoccuparsi delle complessità legate alle rappresentazioni che derivano da tecniche di codifica o sintassi particolari. Inoltre EARMARK fornisce un framework di lavoro completo: le operazioni necessarie per realizzare la conversione, come ad esempio ricerche, manipolazioni delle strutture, ecc., possono servirsi delle tecnologie e degli strumenti del Semantic Web, che permettono di ottenere risultati simili a quelli ottenibili con gli strumenti XML.

I due passi che realizzano la conversione vengono implementati da due

applicazioni distinte: ROCCO (Read, Overhaul, Convert, Classify, Organize) permette di trasformare documenti XML contenenti overlapping in documenti EARMARK, e FRETTA (From Earmark To Tag) consente di realizzare il passo inverso, convertendo documenti EARMARK in documenti XML. In questo modo il framework EARMARK permette di realizzare la conversione fra diversi formati XML.

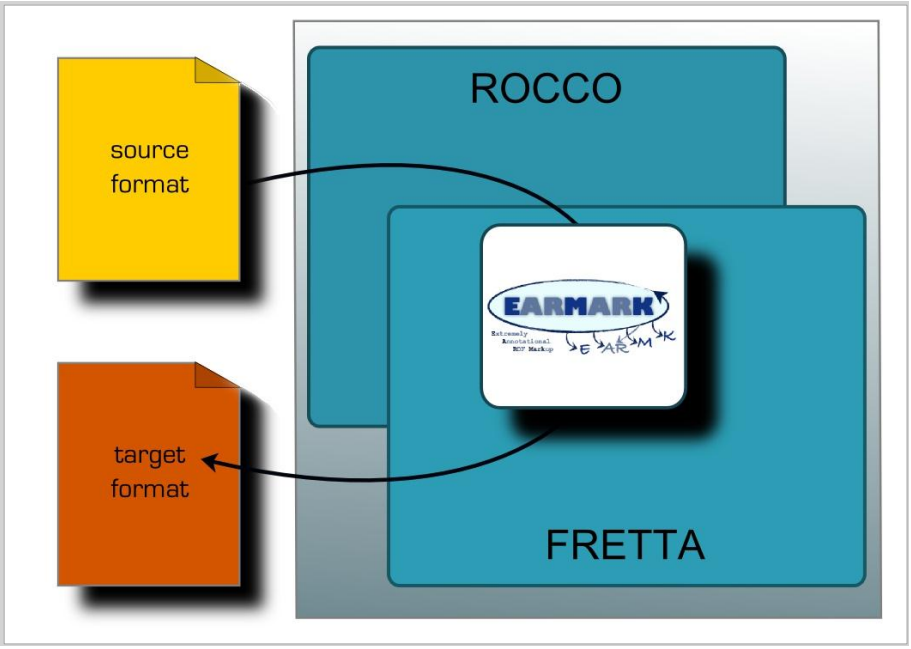

Figura 3.6. Il framework permette di realizzare la conversione fra formati XML differenti: ROCCO converte documenti XML in documenti EARMARK, e FRETTA trasforma documenti EARMARK in formati XML.

3.3 FRETTA

FRETTA è quella parte del framework EARMARK che si occupa di convertire documenti EARMARK in documenti XML. In particolare FRETTA fornisce gli strumenti necessari per convertire quelle caratteristiche complesse di EARMARK che non sono direttamente linearizzabili in XML. Il problema della conversione fra formati viene scomposto in due fasi che permettono di affrontare gli aspetti sintattici e quelli semantici legati al formato oggetto della conversione in maniera separata:

1. per prima cosa FRETTA si occupa di convertire i costrutti EARMARK non direttamente esprimibili in XML secondo le indicazioni specificate dall'utente (ad esempio l'utente potrà dare indicazione di gestire una specifica situazione di overlap all'interno del documento da convertire attraverso una particolare tecnica di codifica)
2. in secondo luogo viene modificata la struttura del documento in funzione della semantica dei formati indicata dall'utente. Poniamo ad esempio che si intenda convertire un documento EARMARK contenente informazioni di change tracking nel formato OASIS Open Document [WB11]: in questo caso FRETTA deve interpretare secondo una certa semantica (es: change tracking) relativa a un certo formato (es: ODT) specifiche parti del documento EARMARK (sia elementi strutturali del documento che metadati con le informazioni di modifica ad essi relativi), e quindi eseguire le modifiche definite per quella particolare interpretazione (ad esempio iniettando alcuni di questi metadati all'interno della struttura del documento in specifici elementi di markup).

Attraverso questi due passi il FRETTA fornisce un meccanismo generale per la conversione di documenti EARMARK in formati XML.

Struttura di FRETTA

La conversione di documenti EARMARK in documenti XML realizzata da FRETTA si articola in quattro passi, ognuno dei quali prende in input un documento, svolge su di esso una o più operazioni, producendo in output una nuova versione modificata del documento. Le operazioni di conversione svolte dai diversi passi vengono svolte su documenti EARMARK in quanto, come detto in precedenza, questo formato permette di gestire in maniera esplicita le feature complesse legate alle situazioni di overlap. La linearizzazione nel formato di destinazione viene realizzata solamente all'ultimo passo di FRETTA, dopo che il documento è stato convertito in modo da rispettare gli aspetti sintattici e i vincoli strutturali del formato di arrivo. In figura 3.7 è rappresentata la struttura generale di FRETTA, mentre in tabella 3.1 è riportata una sintesi delle fasi in cui si articola la conversione, con una breve descrizione delle operazioni svolte in ogni passo.

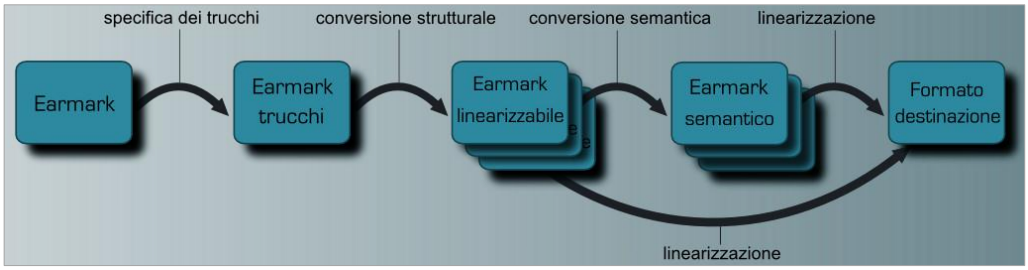

Figura 3.7 Struttura di FRETTA

PASSO		INPUT	OPERAZIONE	OUTPUT
I.	SPECIFICA DEI TRUCCHI	Documento EARMARK	FRETTA fornisce un insieme di trucchi per esprimere le feature di EARMARK non direttamente esprimibili in XML. L'utente può specificare di convertire una o più parti del documento (markup item e range) con i trucchi gestiti da FRETTA.	**EARMARK + TRUCCHI** Un documento EARMARK in cui sono specificati i trucchi da utilizzare per la conversione.
II.	CONVERSIONE STRUTTURALE	EARMARK + TRUCCHI	FRETTA effettua un'analisi del documento e ne modifica la struttura secondo i trucchi specificati al passo precedente.	**EARMARK LINEARIZZABILE** Un documento modificato secondo i trucchi indicati. Se l'utente ha correttamente specificato i trucchi per convertire le parti di documento non esprimibili in XML, il documento in output rappresenta una versione linearizzabile del documento di partenza.
III.	CONVERSIONE SEMANTICA	EARMARK LINEARIZZABILE	L'utente può indicare a FRETTA di interpretare e convertire specifiche parti del documento EARMARK secondo una semantica relativa a un certo formato.	**EARMARK LINEARIZZABILE SEMANTICO** Un documento EARMARK LINEARIZZABILE modificato secondo la semantica del formato specificato dall'utente.
IV.	LINEARIZZAZIONE XML	2 casi: • EARMARK LINEARIZZABILE • EARMARK LINEARIZZABILE SEMANTICO	Il documento in input viene linearizzato in un documento XML.	**XML** Questa operazione produce una versione XML in buona forma del documento di partenza, oppure fornisce un messaggio di errore con la descrizione dei problemi occorsi durante la conversione.

Tabella 3.1. Sintesi delle fasi che realizzano la conversione

3.3.1 Passo I: specifica dei trucchi

Poiché l'espressività di EARMARK è superiore a quella di XML (e in generale dei linguaggi di markup embedded), l'operazione di conversione di documenti EARMARK in XML non sempre è possibile in maniera diretta. Poiché i principali approcci XML propongono di aggirare questo limite espressivo rappresentando le strutture in overlap in altri modi (con elementi vuoti che indicano i limiti di un altro elemento, scomponento gli elementi in frammenti che si annidano correttamente nella gerarchia principale, con riferimenti indiretti, ecc.), d'ora in poi faremo riferimento a queste tecniche di codifica chiamandole semplicemente trucchi sintattici.

Per realizzare la linearizzazione delle situazioni di overlap FRETTA prevede che durante questa fase l'utente specifichi come debba venire gestito ognuno di questi casi. La versione corrente di FRETTA implementa:

- milestone
- fragmentation
- stand-off
- dominance
- sameAs
- copyOf

Milestone, *fragmentation* e *stand-off* sono tre delle quattro principali tecniche descritte in letteratura e analizzate nel capitolo precedente (vedi sezione 2.3.1). *SameAs* e *copyOf* sono soluzioni meno diffuse delle precedenti e per questo non sono state presentate al capitolo precedente. Queste tecniche permettono di risolvere le situazioni in cui elementi presentato contenuto ripetuto introducendo specifici attributi che fanno riferimento al contenuto condiviso (vedi sezione 3.3.1.1). *Dominance* è un trucco introdotto da FRETTA che permette di gestire quei casi in cui le situazioni di overlap possono essere risolte semplicemente introducendo relazioni di dominanza non presenti nel documento originario (maggiori dettagli nella sezione 3.3.1.2).

Inoltre l'utente ha anche la possibilità di specificare all'applicazione di ignorare alcune parti del documento: in questo caso le parti indicate verranno escluse

dalla conversione e non saranno presenti nel documento XML prodotto.

Oltre ai problemi legati all'overlap EARMARK presenta altre caratteristiche complesse che non sono direttamente linearizzabili in XML: nella sezione 3.3.1.3 chiariremo come possano essere gestite con i mecanismi attualmente implementati da FRETTA.

3.3.1.1 CopyOf e sameAs:

Un'altra feature esprimibile in EARMARK ma non direttamente linearizzabile in XML sono elementi che contengono elementi ripetuti. Ad esempio in figura 3.8, (a) descrive un elemento A che contiene l'elemento child1, l'elemento child2 e nuovamente l'elemento child1, mentre (b) è una rappresentazione esplicita di tale documento in EARMARK.

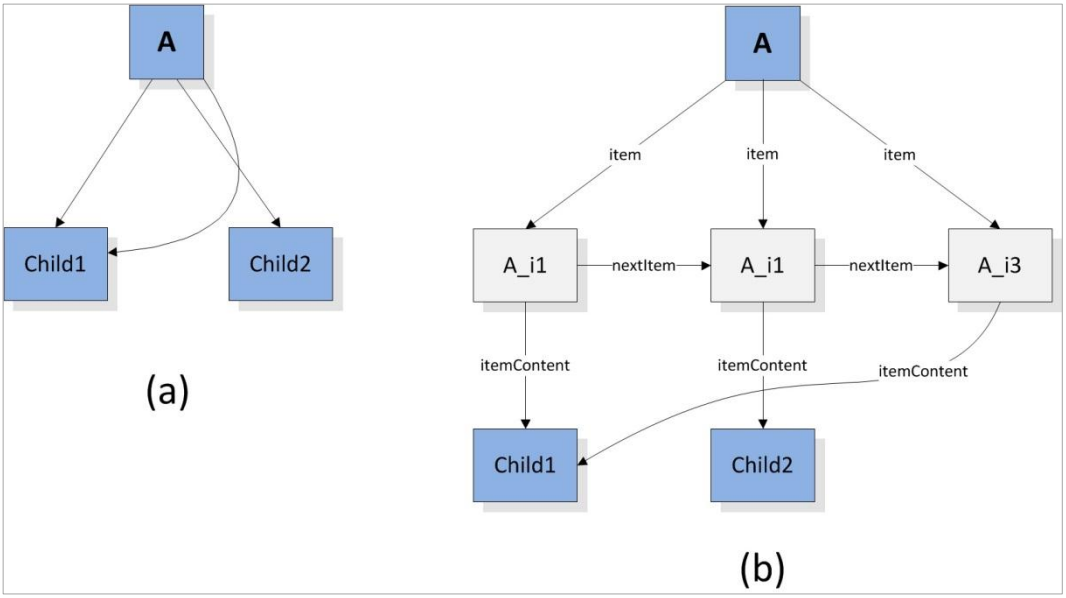

Figura 3.8 Un esempio di elemento con contenuto ripetuto.

TEI P5 propone di esprimere gli elementi ripetuti attraverso *virtual elements,* ossia elementi che non sono presenti in maniera esplicita all'interno del documento, ma la cui presenza può essere inferita in funzione della codifica utilizzata.

Per rappresentare questo tipo di elementi si utilizza un meccanismo di linking via attributi: l'elemento cui si vuole fare riferimento deve contenere un attributo "id" che lo identifichi in maniera univoca, mentre i virtual elements origine dei link utilizzano uno di questi due attributi:

- *sameAs* punta ad un elemento che è identico all'elemento corrente
- *copyOf* punta ad un elemento di cui l'elemento corrente è una copia

L'esempio della figura precedente può venire espresso in XML con uno di queste due tecniche, come mostrato in seguito:

```
<A>
  <child1 xml:id="d840404">...content here...</child1>
  <child2>...content here...</child2>
  <child1 sameAs="d840404">...content here...</child1>
</A>
```

soluzione 1 – una codifica del'esempio in figura 3.8 con il trucco *sameAs*

```
<A>
  <child1 xml:id="d840404">...content here...</child1>
  <child2>...content here...</child2>
  <child1 copyOf="d840404"/>
</A>
```

soluzione 2 – una codifica dell'esempio 3.8 con il trucco *copyOf*

3.3.1.2 Dominance

Oltre alle soluzioni presentate al paragrafo precedente, in alcuni casi il problema dell'overlap può venire risolto introducendo semplicemente relazioni di dominanza non presenti nel documento originale. Consideriamo per esempio il documento EARMARK in figura 3.9 (in cui i rettangoli rappresentano elementi e i cerchi rappresentano range):

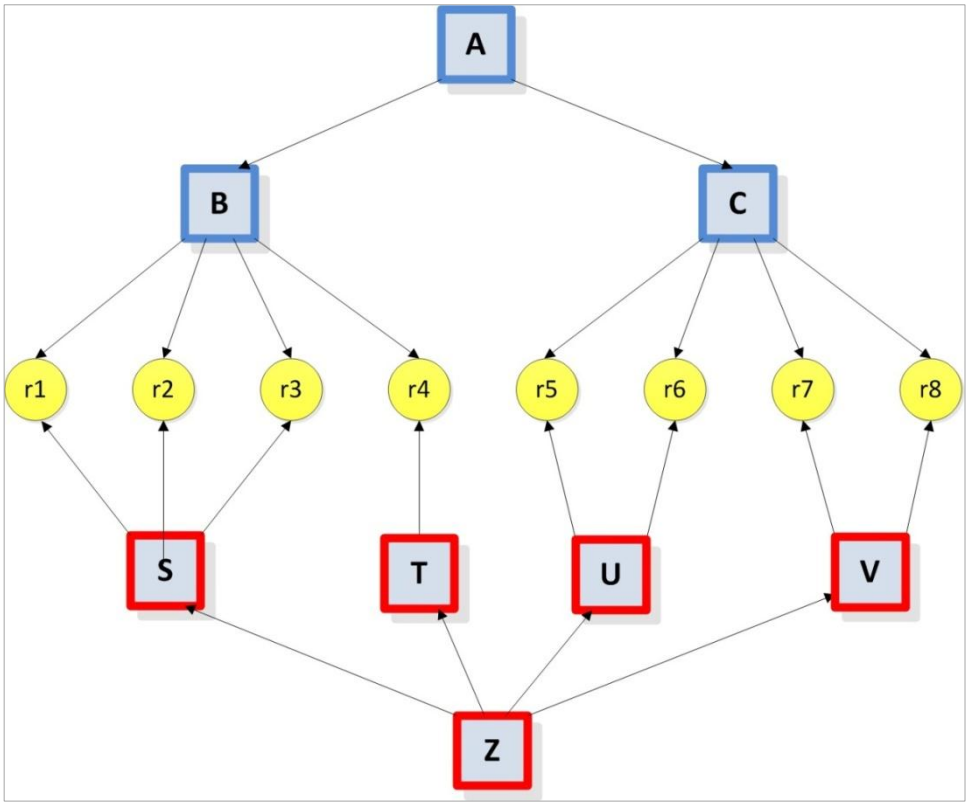

Figura 3.9 Un documento EARMARK contenente due gerarchie in overlap sugli stessi range

Volendo ottenere un documento linearizzabile in XML, si può ipotizzare una conversione che produca un documento come rappresentato in figura 3.10.

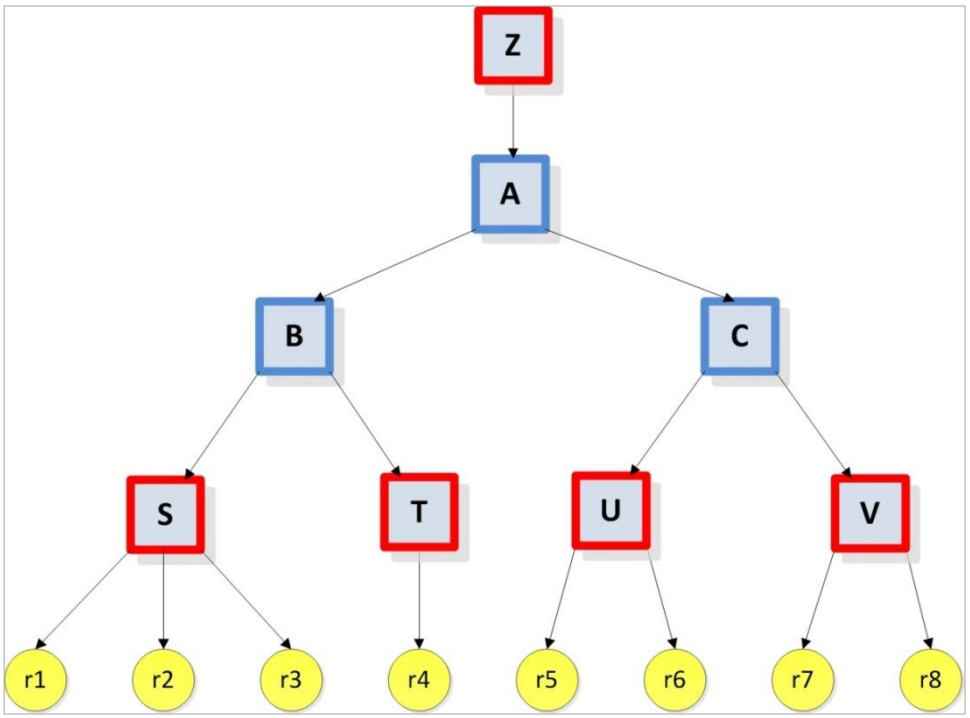

Figura 3.10

Questa trasformazione ha preservato le relazioni di dominanza all'interno della stessa gerarchia, ma ha aggiunto dominanza fra i nodi delle due gerarchie: l'elemento Z continua infatti a dominare gli elementi S, T, U, V, e questi rispettivamente dominano i range che dominavano prima della trasformazione; allo stesso modo A continua a dominare gli elementi B e C, e anche questi continuano a dominare i range che dominavano in precedenza.

Prima della trasformazione però gli elementi non-foglia delle due gerarchie non erano legati da relazioni di dominanza, mentre in seguito gli elementi A, B, e C sono dominati dall'elemento Z, gli elementi S e T sono dominati da A e B e gli elementi U e V sono dominati da A e C.

Utilizzando lo stesso principio si sarebbe potuto trasformare il documento di partenza in un'altra maniera, ottenendo un documento uguale a quello in fig.3.10 con la sola differenza di avere invertiti gli elementi Z e A.

3.3.1.3 Altre caratteristiche di EARMARK non direttamente linearizzabili

Le tecniche sopra elencate permettono di gestire situazioni di overlap altrimenti non esprimibili in XML. Inoltre ci sono altri aspetti complessi di EARMARK da prendere in considerazione: in tabella 3.2 vengono presentate queste feature e per ognuna viene indicato come può essere gestita da FRETTA.

FEATURE COMPLESSE	TIPO DI GESTIONE
Ciclicità/aciclicità del grafo	FRETTA gestisce solo grafi aciclici. Si assume di avere in input solo documenti senza cicli.
Attributi strutturati	Non sono esprimibili in XML. Soluzioni possibili: specificare di non gestire questi attributi; FRETTA può essere esteso da un nuovo trucco che converta gli attributi strutturati in attributi semplici (ad esempio concatenando il nome dell'attributo con il nome del padre).
Attributi con stesso general identifier (relativi allo stesso elemento)	Non sono permessi in XML. Una soluzione possibile è specificare di non convertire l'elemento. Oppure introdurre un nuovo trucco che ne modifichi il general identifier (ad esempio se si tratta di contenuto ordinato espresso attraverso Bag, giustappongo al general-id un numero progressivo).
Gestione docuversi	E' possibile specificare quali docuversi (o parte di essi) andrà a costituire il contenuto del documento, quale il contenuto degli attributi e quale, se presente, dovrà venire ignorato.
Reverse order range	Di per sè questo tipo di range non crea problemi di conversione. (verificare comunque che non si verifichi che più range condividano parte di docuversi, nè situazioni di range condivisi)
Reverse order element	Non sono direttamente esprimibili in XML: l'utente può indicare una gestione specifica per tali elementi (ad esempio con gli stessi trucchi utilizzati per gli elementi che contengono elementi non contigui).
Edge repeatability (elementi che contengono ripetizioni di elementi o di range)	Questa situazione non è direttamente esprimibile in XML. Per risolvere la ripetizione di elementi di markup FRETTA fornisce i trucchi copyOf e sameAs (vedi sezione 3.3.1.1). Per gestire la ripetizione di range è necessario introdurre un nuovo trucco (che ad esempio introduca nel documento un nuovo elemento che contenga il range ripetuto, e che utilizzi meccanismi di linking per fare riferimento ad esso dove si intende esprimere ripetizioni del contenuto testuale cui fa riferimento)
Più range che condividono parte di docuversi	Sono range diversi che rappresentano stringhe diverse, quindi FRETTA nell'ultima fase della conversione genera nodi di testo diversi per ogni range.
Range e/o elemnti condivisi da più padri	Non sono sempre esprimibili in XML: qualora non fosse possibile l'utente deve indicare per essi una specifica gestione attraverso trucchi.
Elementi che contengono elementi non contigui	Non sono esprimibili in XML: l'utente deve indicare trucchi specifici per esprimere questi casi.
Elementi che contengono range non contigui	Non sono esprimibili in XML: l'utente deve indicare trucchi di gestione specifici per questi casi.

Tabella 3.2.

3.3.2 Passo II: conversione strutturale

In questa fase FRETTA effettua un'analisi del documento, ne modifica la struttura secondo le indicazioni specificate al passo precedente e genera una nuova versione del documento. Il contenuto testuale del documento ottenuto è lo stesso di quello del documento di partenza, escluse ovviamente le parti di documento che l'utente ha indicato di non convertire e che quindi sono state eliminate. Le modifiche effettuate in questo passo riguardano invece la struttura del documento: le tecniche indicate dall'utente impongono di adattare specifici elementi ed attributi per esprimere la semantica dell'overlap in maniera implicita, alterando quindi la struttura gerarchica del documento.

Se l'utente ha individuato le parti di documento non esprimibili in XML e ha correttamente specificato come gestirle, il documento in output rappresenta una versione linearizzabile del documento di partenza. A questo punto l'utente può decidere se procedere direttamente alla linearizzazione, oppure se eseguire un'ulteriore passo di conversione (vedi sez. 3.3.3).

3.3.3 Passo III: conversione semantica

Nella terza fase FRETTA si occupa di interpretare secondo una certa semantica relativa a un certo formato specifiche parti del documento EARMARK, e quindi eseguire le modifiche definite per quella particolare interpretazione: se in questo passo l'utente specifica di utilizzare la semantica relativa ad un particolare formato (al momento è stata implementata la semantica del formato TEI [SB05]), FRETTA utilizzerà tale semantica per interpretare specifiche parti del documento EARMARK (sia elementi strutturali del documento che metadati ad essi relativi), e quindi eseguire le modifiche che l'utente ha indicato per quella particolare interpretazione (ad esempio iniettando alcuni di questi metadati all'interno della struttura del documento in specifici attributi ed elementi di markup).

3.3.4 Passo IV: Linearizzazione XML

In questo passo FRETTA prende in input un documento EARMARK e ne produce una versione linearizzata in XML. Il documento in input può essere quello generato nel passo II (EARMARK linearizzabile) o nel passo III (EARMARK semantico): l'utente infatti può decidere se eseguire solo la conversione strutturale, che consente di convertire le parti di documento non esprimibili in XML in strutture linearizzabili, oppure svolgere un ulteriore passo che interpreti e converta parti di documento secondo la semantica di uno specifico formato (vedi sez. 3.3.3).

Il risultato di questa fase è il prodotto finale di tutto il processo di conversione eseguito dall'applicazione, e dovrebbe risultare in una versione XML in buona forma del documento in input. In questo passo però non viene eseguito alcun tipo di controllo sul documento: l'utente è infatti responsabile di specificare come deve essere eseguita la conversione, e i passi precedenti di FRETTA devono garantire la corretta esecuzione delle modifiche specificate dall'utente. Questo però non assicura che la conversione abbia successo: è responsabilità dell'utente individuare le parti di documento non esprimibili in XML e specificare a FRETTA le corrette informazioni di gestione.

Consideriamo ad esempio un documento EARMARK contenente feature complesse quali attributi strutturati e situazioni di overlap. Perché la conversione abbia successo l'utente deve:

1. individuare gli attributi strutturati e indicare al FRETTA di tralasciarli, escludendoli quindi dal documento finale, oppure estendere l'applicazione introducendo una nuova tecnica di codifica e specificare di esprimere questi attributi con questa nuova tecnica.
2. individuare le situazioni di overlap e per ognuna specificare come debba venire gestita. Non tutti le tecniche di codifica consentono però di esprimere tutte le situazioni di overlap possibili: ad esempio milestone non è in grado di esprimere elementi discontinui.

Se l'utente non specifica in maniera corretta come gestire queste situazioni, l'applicazione non è in grado di convertire il documento: durante questo passo FRETTA fornirà all'utente messaggi di errore che descrivono i problemi occorsi durante la conversione.

Capitolo 4

FRETTA:

STRUTTURA DELL'APPLICAZIONE

Come detto nei capitoli precedenti l'obiettivo di questa tesi è la realizzazione di un meccanismo di conversione di documenti EARMARK contenenti strutture in overlap in documenti XML. Dopo avere introdotto il contesto scientifico all'interno del quale è stata realizzata la nostra soluzione e avere evidenziato i principali problemi che sono stati affrontati (capitolo 2), nel capitolo precedente (capitolo 3) abbiamo descritto EARMARK, che rappresenta la principale tecnologia utilizzata nel nostro lavoro, e abbiamo fornito una descrizione generale dell'applicazione realizzata.

In questo capitolo analizzeremo la struttura di FRETTA, presentando in maniera dettagliata le soluzioni sviluppate per realizzare il motore della nostra

applicazione.

L'implementazione di FRETTA che proponiamo è stata sviluppata in Java, ed è strutturata come descritto al capitolo precedente: l'applicazione esegue sequenzialmente i quattro passi principali in cui si articola il processo di conversione (specifica delle tecniche di codifica, conversione strutturale, conversione semantica, linearizzazione). Ogni passo prende in input un documento, svolge su di esso una o più operazioni, producendo in output una nuova versione modificata del documento. Le operazioni di conversione effettuate dai diversi passi vengono svolte su documenti EARMARK in quanto questo formato permette di gestire in maniera esplicita le feature complesse legate alle situazioni di overlap. La linearizzazione nel formato di destinazione viene realizzata solamente all'ultimo passo di FRETTA, dopo che il documento è stato convertito in modo da rispettare i vincoli del formato di arrivo.

Per gestire i documenti EARMARK utilizzati durante la conversione sono state utilizzate le EARMARK API (vedi sez. 3.1.2), una libreria Java che permette di creare, validare e manipolare le strutture dati di EARMARK.

Per chiarire le operazioni svolte dalle singole fasi vengono mostrate le operazioni necessarie per realizzare la conversione nel formato TEI P5 [SB05] del frammento proposto nell’esempio 2.1, di cui in figura 3.5 è stata rappresentata una codifica in EARMARK.

4.1 Specifica dei trucchi

Come detto in precedenza EARMARK permette di descrivere documenti contenenti strutture in overlap che non possono essere direttamente linearizzate in XML. Gli elementi in overlap non si annidano correttamente, per cui non rispettano l'organizzazione gerarchica imposta dai linguaggi basati sul modello ad albero come XML. L'unico modo di rappresentare gerarchie in overlap è di utilizzare trucchi sintattici: queste tecniche prescrivono di individuare una gerarchia principale che costituisce un albero e rappresentatarla direttamente in XML, e di nascondere tutti gli elementi rimanenti all'interno della gerarchia principale in un'altra forma: elementi

vuoti che indicano i limiti di un elemento nel caso delle *milestone*, elementi divisi in frammenti per la *fragmentation*, ecc.

Il meccanismo di conversione realizzato da FRETTA richiede che l'utente identifichi le situazioni di overlap e fornisca all'applicazione istruzioni sulle tecniche da utilizzare per esprimere questi casi nel formato di arrivo. Per specificare queste informazioni è stato definito un formato XML che permette di associare ad ogni elemento il relativo trucco sintattico (vedi sez. 4.1.2).

I trucchi sintattici di cui dispone FRETTA sono definiti in una semplice ontologia OWL, che contiene le classi che rappresentano i trucchi utilizzabili durante la conversione (vedi sezione 4.1.1).

Un altro aspetto importante da considerare durante questa fase è il document order: lo stesso contenuto documentale può infatti essere organizzato in ordini differenti e incompatili all'interno delle diverse gerarchie descritte dai documenti EARMARK. In XML invece l'ordinamento è definito in maniera univoca dalla disposizione dei caratteri del contenuto testuale del documento. L'unico modo di rappresentare strutture con ordinamenti incompatibili in XML è quindi quello di definire il document order dei nodi foglia dell'albero (nodi testuali ed elementi vuoti) ed esprimere tutti gli elementi (sia quelli appartenenti alla gerarchia principale che quelli delle gerarchie secondarie) in funzione di tale ordinamento. Perché la conversione abbia successo è necessario specificare i trucchi sintattici da utilizzare per codificare le situazioni che non rispettano tale ordinamento (ad esempio elementi discontinui o interrotti). Per maggiori informazioni sulla gestione del document order consultare la sez. 4.1.3.

E' importante osservare che non tutti i trucchi hanno la stesso potere espressivo: ad esempio il trucco milestone non è in grado di rappresentare elementi discontinui o con contenuto in un ordine diverso da quello definito per il documento.

4.1.1 Ontologia di gestione dei trucchi

L'approccio di EARMARK suggerisce di modellare i documenti come collezioni di frammenti di testo indirizzabili, e di associare a tale contenuto testuale delle asserzioni OWL che ne descrivano le caratteristiche strutturali. Allo stesso

modo è possibile aggiungere in qualsiasi momento annotazioni semantiche attraverso ulteriori asserzioni, e anche queste possono essere in overlap con il contenuto precedente. FRETTA sfrutta quest'ultima possibilità per descrivere tutte le informazioni rilevanti per realizzare il processo di conversione: per ogni trucco specificato dall'utente vengono ad esempio inserite all'interno del documento EARMARK oggetto della conversione specifiche asserzioni RDF che mettono in relazione l'elemento da convertire con il trucco indicato.

Le annotazioni che riguardano la specifica dei trucchi vengono modellate su una semplice ontologia OWL 2 che definisce una classe per ogni trucco implementato[10].

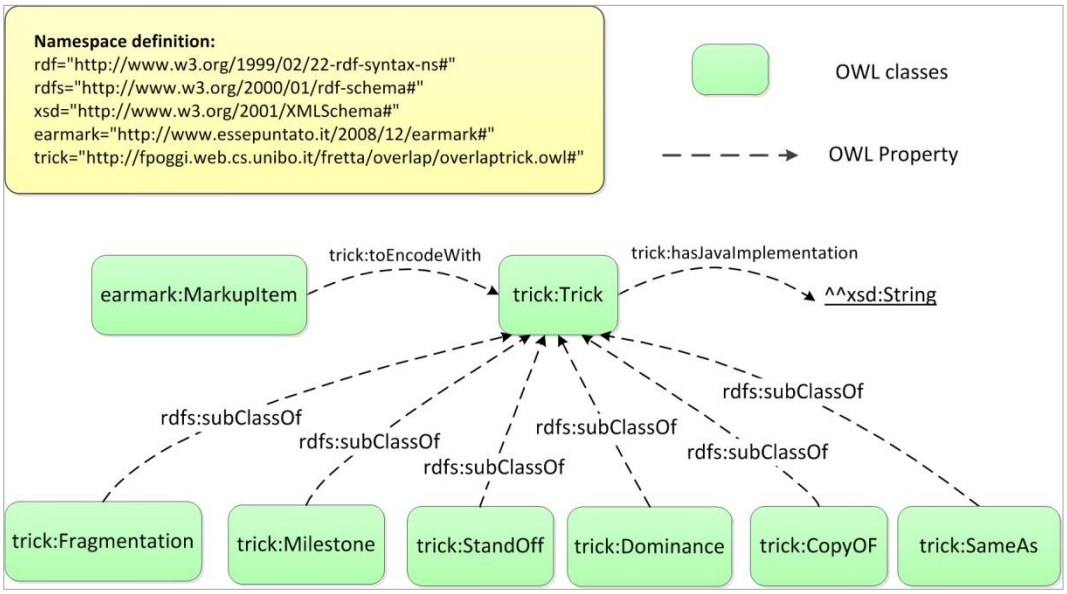

Figura 4.1 Una rappresentazione grafica dell'ontologia dei trucchi.

La classe trick:Trick descrive il concetto di trucco sintattico. Questa classe viene specificata in sei sottoclassi che rappresentano i trucchi implementati da FRETTA (*milestone*, *fragmentation*, s*tand-off, dominance, copyOf* e *sameAs*). Sono state definite due proprietà principali: *trick:ToEncodeWith* è una Object Property che mette in relazione elementi EARMARK con i trucchi da utilizzare per la conversione, e *trick:hasJavaImplementation* è una Datatype Property che collega un trucco con valore di tipo stringa che rappresenta la classe Java che implementa il trucco.

[10] L'ontologia dei trucchi è disponibile all'indirizzo http://fpoggi.web.cs.unibo.it/fretta/overlap/overlaptrick.owl

Quest'ultima proprietà viene utilizzata per implementare il meccanismo di estensioni di FRETTA che verrà descritto più approfonditamente in seguito (vedi sez. 4.5).

4.1.2 Formato di specifica dei trucchi

I trucchi da utilizzare durante la conversione devono essere indicati attraverso uno specifico formato XML: i file di specifica dei trucchi sono costituiti da un insieme di regole che associano una parte di documento al trucco che deve essere utilizzato per realizzarne la conversione. L'utente può dare indicazione a FRETTA di utilizzare un determinato trucco per un elemento particolare, per tutti gli elementi con un certo general identifier o per gli elementi che appartengono a uno specifico namespace. In esempio 4.1 forniamo un esempio di un file di specifica dei trucchi relativo al documento EARMARK che rappresenta una codifica del frammento estratto da *La Mort d'Agrippine* discusso nei capitoli precedenti (è stata fornita una rappresentazione grafica del documento in figura 3.5).

```
<?xml version="1.0"?>

<!DOCTYPE trickspec [
   <!ENTITY trick "http://fpoggi.web.cs.unibo.it/fretta/overlap/overlaptrick.owl#">
   <!ENTITY exampleCyrano
      "http://fpoggi.web.cs.unibo.it/fretta/example/Cyrano.owl#">
]>

<trickspec xmlns="http://fpoggi.web.cs.unibo.it/fretta/example/trickSpecification"
   xmlns:trick="http://fpoggi.web.cs.unibo.it/fretta/overlap/overlaptrick.owl#"
   xmlns:exampleCyrano="http://fpoggi.web.cs.unibo.it/fretta/example/Cyrano.owl#"

   <trick>
      <refID>&exampleCyrano;#sp1</refID>
      <type>&trick;Milestone</type>
   </trick>
   <trick>
      <refID>&exampleCyrano;#sp2</refID>
      <type>&trick;Milestone</type>
   </trick>
   <trick>
      <refID>&exampleCyrano;#sp3</refID>
      <type>&trick;Milestone</type>
   </trick>
   <trick>
      <refID>&exampleCyrano;#sp4</refID>
      <type>&trick;Fragmentation</type>
   </trick>
   <trick>
```

```
      <refGID>&exampleCyrano;#speaker</refID>
      <type>&trick;Dominance</type>
   </trick>

</trickspec>
```

Esempio 4.1: un file di specifica dei trucchi relativo al documento presentato in figura 3.5.

Ogni elemento `<trick>` rappresenta una singola istanza di regola di specifica dei trucchi. All'interno di questi elementi vengono definiti le parti di documento da codificare (attraverso uno fra gli elementi `<refID>`, `<refGID>` o `<refNS>`) e il trucco da utilizzare (attraverso l'elemento `<type>`). Gli elementi `<refID>`, `<refGID>` e `<refNS>` vengono utilizzati per associare un trucco rispettivamente ad un elemento (fornendone l'URI), ad un insieme di elementi con un particolare general identifier o ad elementi che appertengono ad un certo namespace. Il trucco da utilizzare viene indicato specificando l'URI della rispettiva classe all'interno dell'ontologia di specifica dei trucchi presentata nella sezione 4.1.1.

Nell'esempio precedente si è scelto di rappresentare con trucchi la gerarchia dei dialoghi: per ottenere una corretta linearizzazione del documento è necessario gestire con trucchi tutti gli elementi `<sp>` e `<speaker>`. In questo caso è stato indicato di utilizzare milestone per tutti gli elementi `<sp>` tranne l'ultimo che deve venire codificato con *fragmentation*, mentre per tutti gli elementi `<speaker>` verrà utilizzato il trucco *dominance*.

Come detto precedentemente il documento EARMARK di partenza viene arricchito da asserzioni RDF che specificano i trucchi definiti dall'utente in questa fase. La figura 4.2 rappresenta il documento EARMARK prodotto da FRETTA in questa fase della conversione. Per chiarezza proponiamo un estratto relativo al sottoalbero dominato dall'ultimo elemento `<sp>`.

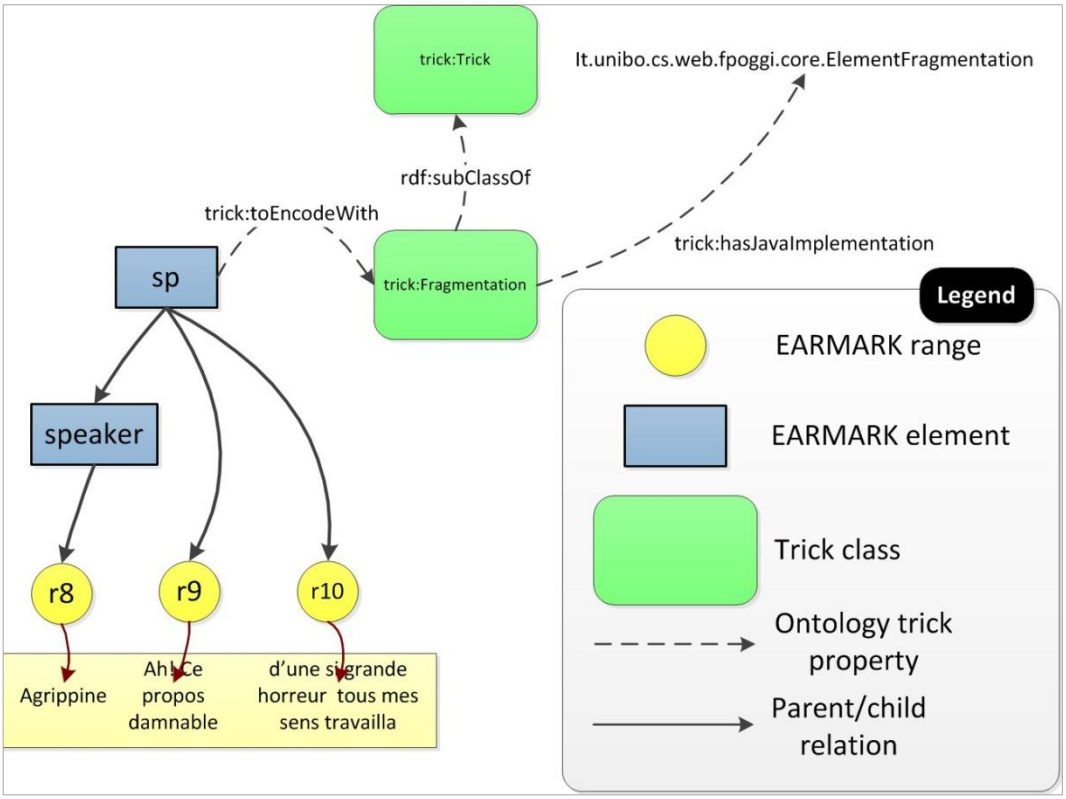

Figura 4.2 Una rappresentazione della sottogerarchia radicata nell'ultimo element `<sp>`[11]

4.1.3 Document order

Come detto in precedenza, l'unico modo di rappresentare strutture con ordinamenti incompatibili in XML è di definire un ordinamento dei nodi foglia dell'albero (nodi testuali ed elementi vuoti) valido per tutte le gerarchie del documento ed esprimere tutti gli elementi (sia quelli appartenenti alla gerarchia principale che quelli delle gerarchie secondarie) in funzione di tale ordinamento.

La gestione del document order di FRETTA è la seguente: qualora non venga specificato nessun document order, l'applicazione esegue un semplice algoritmo che calcola un ordinamento dei nodi foglia. L'algoritmo realizza una visita pre-order di tutto il grafo e restituisce un document order in cui ogni foglia si trova nella prima posizione in cui è stata incontrata durante la visita. Alternativamente l'utente può indicare a FRETTA un diverso ordinamento attraverso uno specifico formato XML: in questo caso l'utente deve assicurarsi di inserire in questo ordinamento tutti i nodi

[11] Il meccanismo che arricchisce il documento EARMARK con asserzioni descritto in queste pagine fa uso di punning, una semplice forma di meta-modeling introdotta da OWL 2.

foglia dominati dagli elementi che devono essere convertiti. Ad esempio si potrebbe desiderare ottenere una versione del testo precedente in cui le coppie personaggio-battuta vengano invertite di ordine: il frammento seguente mostra una specifica di document order che permette di ottenere, come richiesto, una linearizzazione in cui il nome dei personaggi seguono il testo delle battute da essi recitate.

```
<?xml version="1.0"?>

<documentOrder
   xmlns="http://fpoggi.web.cs.unibo.it/fretta/example/documentOrder"
   xmlns:exampleCyrano="http://fpoggi.web.cs.unibo.it/fretta/example/
      Cyrano.owl#"

   <leafNode>&exampleCyrano;#r2</leafNode>
   <leafNode>&exampleCyrano;#r1</leafNode>
   <leafNode>&exampleCyrano;#r4</leafNode>
   <leafNode>&exampleCyrano;#r3</leafNode>
   <leafNode>&exampleCyrano;#r6</leafNode>
   <leafNode>&exampleCyrano;#r7</leafNode>
   <leafNode>&exampleCyrano;#r8</leafNode>
   <leafNode>&exampleCyrano;#r5</leafNode>
   <leafNode>&exampleCyrano;#r10</leafNode>
   <leafNode>&exampleCyrano;#r9</leafNode>

</documentOrder >
```

Esempio 4.4 Un file di specifica del document order relativo al documento rappresentato in figura 3.5

Il risultato di questo passo è una versione del documento di partenza arricchito da asserzioni RDF che specificano i trucchi definiti dall'utente e il document order da utilizzare per relizzare la conversione.

4.2 Conversione strutturale

Il passo di conversione strutturale si occupa di modificare la struttura del documento EARMARK ottenuto alla fase precedente secondo i trucchi indicati dall'utente generando una nuova versione del documento. Per realizzare questa operazione FRETTA crea un nuovo documento EARMARK e vi inserisce tutti gli elementi per cui non è stato specificato un trucco: tali elementi costituiscono la gerarchia principale del documento XML finale, per cui se l'utente ha specificato le istruzioni di conversione in maniera corretta al passo precedente tale gerarchia potrà venire espressa in XML in maniera esplicita, senza ricorrere a particolari tecniche di

codifica. Per garantire che questa gerarchia sia effettivamente esprimibile in XML, FRETTA analizza il documento ottenuto verificando che rispetti il modello ad albero prescritto da XML.

Gli elementi per cui è stato specificato un trucco appartengono invece alle gerarchie secondarie, e vengono inseriti uno ad uno nel nuovo documento EARMARK utilizzando la tecnica di codifica di ogni trucco. Ad esempio, l'elemento `<sp>` rappresentato in figura 3.5 verrà quindi rappresentato da due diversi frammenti e inserito all'interno della gerarchia principale.

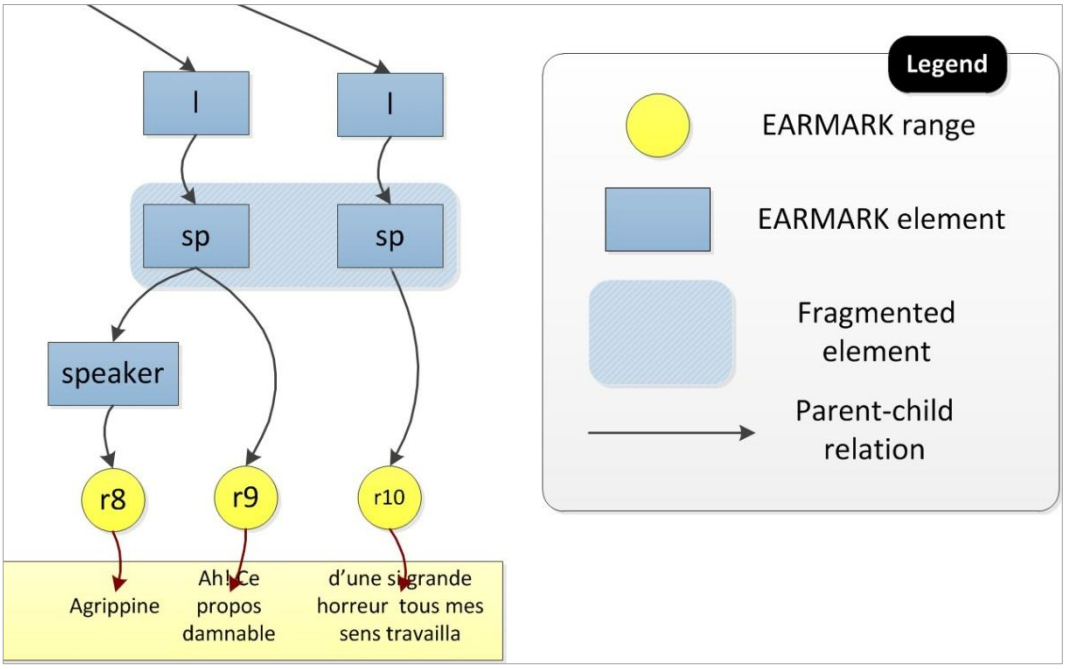

Figura 4.3 Un dettaglio del nuovo documento EARMARK che mostra come sono stati convertiti gli elementi in figura 4.2.

Oltre a generare una nuova versione del documento realizzata secondo le indicazioni specificate dall'utente, FRETTA aggiunge nuove asserzioni al documento originale che descrivono le operazioni svolte durante questo passo: queste indicazioni verranno utilizzate nella fase seguente (vedi sez. 4.3). Considerando ad esempio il caso preso in considerazione precedentemente, il documento EARMARK iniziale verrà arricchito delle asserzioni rappresentate in figura 4.4.

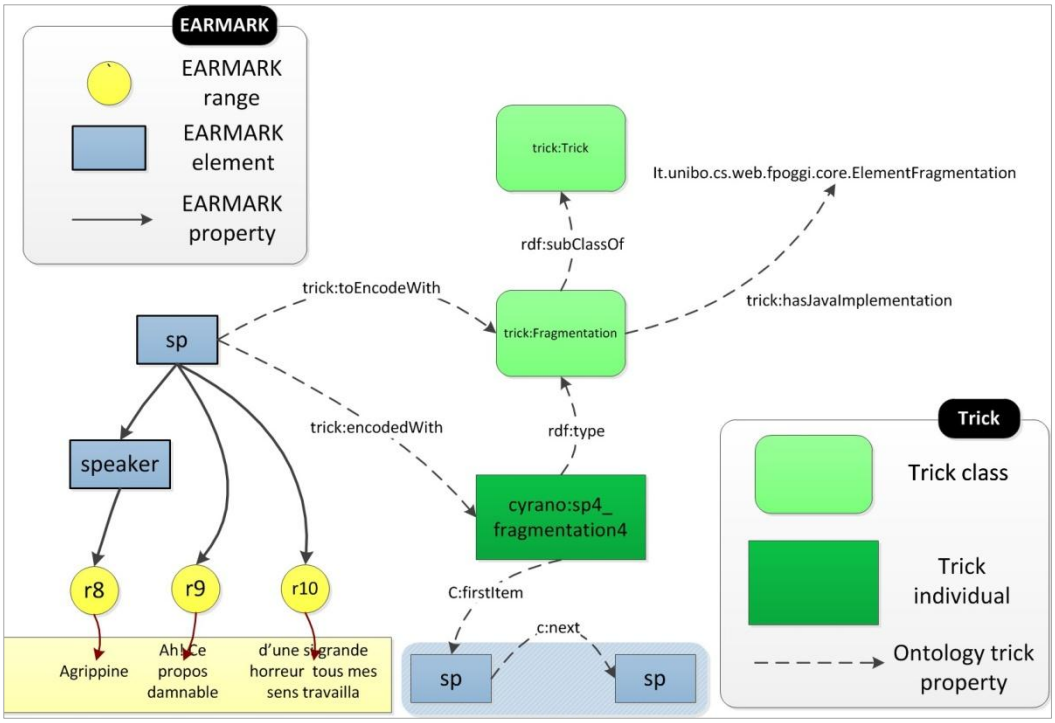

Figura 4.4 Una rappresentazione di un frammento del documento originale arricchito di asserzioni RDF.

Confrontando questa immagine con la figura 4.3 si può osservare che sono state aggiunte alcune asserzioni: la proprietà *trick:encodedWith* permette di esprimere il fatto che l'elemento `<sp>`, a seguito della conversione strutturale, è stato rappresentato da un'istanza specifica del trucco indicato (*cyrano:sp4_fragmentation4*). In particolare, tale istanza è definita come una collezione di elementi ordinati, contenente un primo frammento `<sp>` (*trick:firstItem*) seguito da un altro altro frammento `<sp>` (*trick:next*): questi due elementi sono i due frammenti mostrati in figura 4.3. Guardando le asserzioni in figura 4.4 e risalendo verso l'alto è possibile ricostruire a ritroso le operazioni effettuate durante i primi due passi: durante il passo di conversione strutturale FRETTA ha creato una fragmentation (*cyrano:sp4_fragmentation*) costituita da due nuovi frammenti (i due `<sp>` su fondo azzurro) che permettono di codificare l'elemento `<sp>` del documento originale (*encodedWith*) secondo il trucco specificato dall'utente durante la prima fase della conversione (*toEncodeWith*).

Il meccanismo di conversione implementato per gli altri tucchi è lo stesso di quello presentato per la *fragmentation*. L'unica differenza riguarda il trucco stand-off: in questo caso infatti FRETTA crea un ulteriore documento che contiene tutti gli

elementi da rappresentare in *stand-off*. Poniamo per esempio che nel passo di specifica dei trucchi l'utente decidesse di esprimere con *stand-off* i due elementi `<sp>` e `<speaker>` appartenenti alla gerarchia dei dialoghi rappresentati in figura 4.3 (gli elementi `<sp>` e `<speaker>`): il risultato della conversione di FRETTA è rappresentato nella figura seguente.

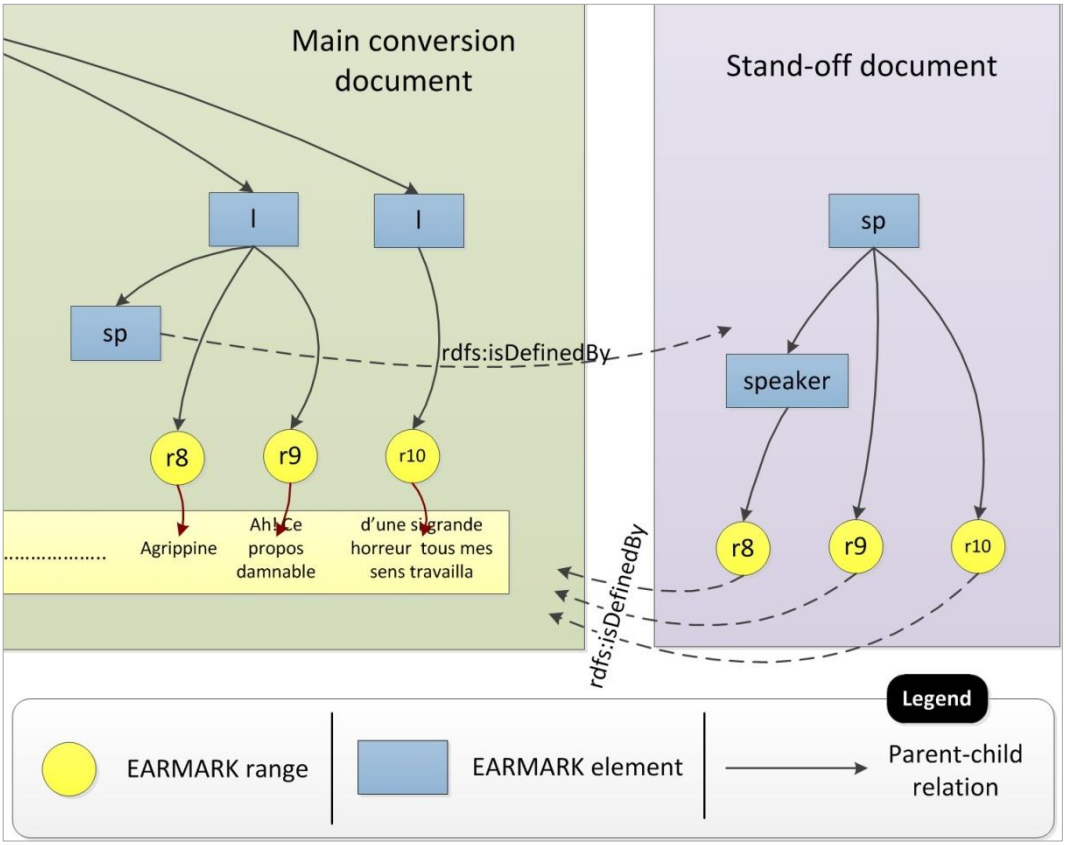

Fig. 4.5 Una rappresentazione del frammento del documento di esempio considerato codificato attrverso il trucco stand-off

Il documento di destra contiene gli elementi in *stand-off* e gli elementi da essi dominati, mentre il documento principale contiene tutti gli elementi che l'utente non ha dichiarato di voler esprimere in *stand-off*. Inoltre ques'ultimo documento contiene l'elemento radice della gerarchia in stand-off svuotato di tutto il suo contenuto, in modo da indicare la posizione che va ad occupare il contenuto in *stand-off* all'interno del documento principale.

Il risultato di questa fase è costituito da tre documenti EARMARK: il primo è il documento fornito in input a FRETTA arricchito di asserzioni RDF che descrivono le operazioni svolte dall'applicazione; il secondo è il documento costituito dalla gerarchia principale e dagli elementi delle gerarchie secondarie espressi attraverso i

trucchi sintattici; infine, se sono stati convertiti elementi con *stand-off*, FRETTA genera un ulteriore documento con le gerarchie degli elementi in *stand-off*: in quest'ultimo caso il secondo ed il terzo documento vengono messi in relazione dalla proprietà *rdfs:isDefinedBy.* A questo punto l'utente ha la possibilità di scegliere se intende effettuare un ulteriore passo di conversione (vedi sez. 4.3), oppure se procedere direttamente alla linearizzazione in XML.

4.3 Conversione semantica

Nella fase di conversione semantica FRETTA deve interpretare secondo una certa semantica relativa a un certo formato specifiche parti del documento EARMARK, e quindi eseguire le modifiche definite per quella particolare interpretazione. FRETTA definisce i meccanismi di conversione relativi al formato TEI P5 [SB05]: se l'utente decide di effettuare questo passo di conversione, FRETTA utilizzerà la semantica relativa al formato TEI per interpretare specifiche parti dei documenti EARMARK prodotte nella fase di conversione strutturale (sia elementi strutturali del documento che metadati ad essi relativi), e quindi rappresenterà gli elementi convertiti con i trucchi specificati dall'utente secondo le tecniche previste da TEI (ad esempio iniettando alcuni di questi metadati all'interno della struttura del documento in specifici elementi di markup, o realizzando con specifici attributi un sistema di linking che colleghi i frammenti che rappresentano un elemento, ecc.).

Considerando l'esempio di conversione portata avanti nel corso di questo capitolo, i frammenti espressi con *fragmentation* durante la fase di conversione strutturale rappresentati in figura 4.5 vengono messi in relazione dagli attributi specificati in TEI, come mostrato in figura 4.6.

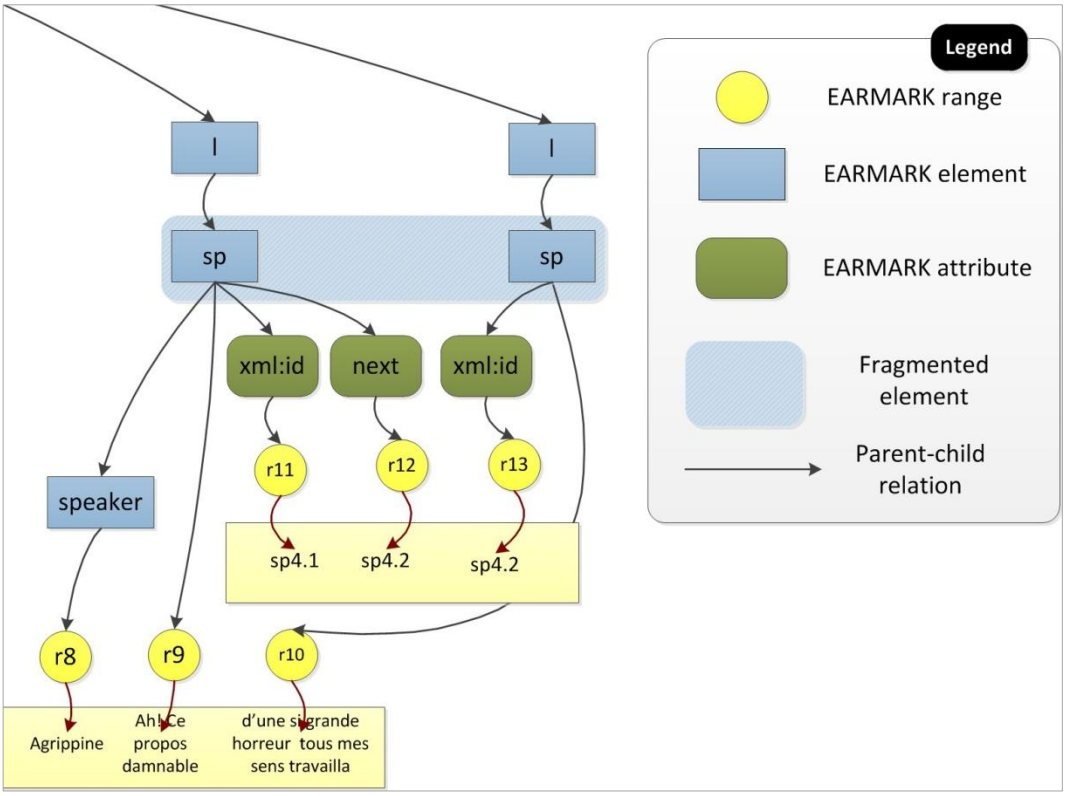

Figura 4.6 Una rappresentazione del documento generato durante la fase di conversione semantica**.**

Come mostrato in figura, il formato TEI prevede di aggiungere ad ogni frammento un attributo "`xml:id`" con l'identificato del frammento, e di collegare i frammenti con un attributo "`next`" che punta all'identificativo del frammento seguente.

Le informazioni necessarie per effettuare le operazioni di conversione previste da questo passo sono definite dalle asserzioni RDF aggiunte al documento in input durante le fasi di specifica dei trucchi e di conversione strutturale (vedi figura 4.2 e figura 4.4). FRETTA può risalire a queste informazioni eseguendo una semplice query SPARQL [PS08], che restituisce per ogni elemento per cui è stato specificato un trucco gli elementi generati durante il passo di conversione strutturale per rappresentarlo: a questo punto l'applicazione risale al trucco specificato ed esegue le le opportune conversioni previste dal formato di arrivo per quel trucco (nel nostro caso da TEI).

Se l'utente ha fornito a FRETTA una specifica dei trucchi corretta, il documento EARMARK prodotto in questa fase rappresenta una versione del documento di partenza convertito in maniera conforme al formato TEI e direttamente linearizzabile in XML (vedi sez. 4.4).

4.4 Linearizzazione

Durante la fase di linearizzazione FRETTA prende in input un documento EARMARK e ne produce una versione linearizzata in XML. Il documento in input può essere quello generato nel passo II (EARMARK linearizzabile) o nel passo III (EARMARK semantico): l'utente infatti può decidere se eseguire solo la conversione strutturale (vedi sez. 4.2), che si occupa di convertire le parti di documento non esprimibili in XML in strutture linearizzabili, oppure svolgere una ulteriore operazione che interpreti e converta parti di documento secondo la semantica di uno specifico formato (vedi sez. 4.3).

Il risultato di questa fase è il prodotto finale di tutto il processo di conversione eseguito da FRETTA, e dovrebbe risultare in una versione XML in buona forma del documento EARMARK in input. In questo passo però non viene eseguita alcuna forma di controllo sul documento: l'utente è infatti responsabile di specificare come deve essere eseguita la conversione, e i passi precedenti di FRETTA devono garantire la corretta esecuzione delle modifiche specificate dall'utente. Questo però non assicura che la conversione abbia successo: è responsabilità dell'utente individuare le parti di documento non esprimibili in XML e specificare all'applicazione le corrette informazioni di gestione.

Per svolgere la linearizzazione FRETTA analizza il documento in input e costruisce l'albero del documento XML di destinazione aggiungendo uno ad uno gli elementi del documento EARMARK. Per realizzare questa operazione vengono utilizzate le API del framework Java che implementano il modello DOM[12]: in questo modo FRETTA non deve eseguire nessuna verifica sulla forma del documento in input poiché le situazioni che violano i vincoli imposti dal modello ad albero di XML vengono riconosciute e segnalate direttamente da questa libreria. Qualora si tenti ad esempio di linearizzare un documento contenente più elementi in overlap sullo stesso elemento verrà sollevata una specifica eccezione Java che potrà essere intercettata da FRETTA e gestita secondo il meccanismo di eccezioni presentato in seguito.

[12] http://www.w3.org/DOM/

4.4.1 Gestione degli errori di conversione

L'utente viene informato dei problemi che sono avvenuti durante il processo di conversione tramite specifici messaggi di errore o di warning. Gli errori generalmente avvengono nell'ultima fase di FRETTA: è infatti solo in questo momento che l'applicazione, dopo avere effettuato le modifiche al documento EARMARK specificate dall'utente, tenta di effettuare la linearizzazione nel formato XML. Se il documento non risulta linearizzabile FRETTA procede in uno dei seguenti modi:

- *errore*: il processo di conversione si interrompe fornendo un messaggio di errore generico, senza terminare il processo.
- *warning generico*: viene generato un messaggio di warning che indica un problema di conversione generico, tutte le parti di documento interessate vengono eliminate e non saranno presenti nel documento generato.
- *warning specifico*: come il passo precedente, ma il messaggio di warning descrive in maniera esaustiva il problema e le parti di documento interessate. In questo modo l'utente può procedere a una successiva conversione, modificando le specifiche di conversione fornite a FRETTA in funzione delle informazioni ricevute.

4.5 Meccanismi di estensione

I passi descritti nelle sezioni precedenti permettono di eseguire la conversione di documenti in EARMARK in documenti XML. Lo sforzo principale di questo lavoro è quello di fornire un meccanismo che permetta di gestire le caratteristiche documentali complesse di EARMARK non direttamente esprimibili in XML. Per risolvere questo problema FRETTA fornisce un insieme di trucchi sintattici (milestone, fragmentation, stand-off markup, copyOf, sameAs, dominance) che permettono di codificare queste situazioni, dando la possibilità all'utente di indicare per ognuna la tecnica di codifica che intende utilizzare. Poiché il progetto all'interno del quale si inserisce FRETTA prevede di utilizzare questa soluzione in contesti

applicativi molto differenti fra loro, il meccanismo di conversione implementato dall'applicazione deve essere il più generale possibile, permettendo di adattarsi ai diversi domini di interesse. Per questo motivo è stata prevista la possibilità di estendere le funzionalità di FRETTA in modo da potere aggiungere in futuro la gestione di ulteriori trucchi sintattici e tecniche di codifica.

Per descrivere i meccanismi di estensioni di FRETTA descriveremo un caso concreto: poniamo che un utente intenda aggiungere all'applicazione la possibilità di utilizzare il trucco *twin documents* descritto nella sezione 2.3.1. Questa operazione può essere realizzata in due passi:

1. Per prima cosa è necessario sviluppare una classe Java che implementi il metodo *resolveTrick()* definito dall'interfaccia *ResolvableTrick*: questo metodo viene invocato durante la fase di conversione strutturale e si occupa di convertire gli elementi per cui è specificato questo trucco secondo la tecnica di codifica dei *twin documents*. Il file .jar contenente le classi sviluppate devono essere poste in una specifica directory nello stesso classpath di FRETTA: a questo punto sarà l'applicazione che si occuperà di caricare dinamicamente questa classe qualora il processo di conversione richieda di invocare il metodo *resolveTrick()* che implementa.
2. In secondo luogo l'utente che intende utilizzare il trucco implementato da questa classe all'interno della propria applicazione deve informare FRETTA del nome del trucco aggiunto e del nome della classe descritta al punto precedente. Per fare questo l'utente estenderà l'ontologia dei trucchi con le informazioni relative al trucco implementato. Durante la fase iniziale della conversione si dovrà quindi fornire a FRETTA questo file OWL che verrà semplicemente caricato all'interno del modello del documento EARMARK, andando così ad estendere l'ontologia dei trucchi descritta nella sezione 3.1.1.

Una volta che sono stati eseguiti questi due passi l'utente ha la possibilità di specificare questo nuovo trucco durante la prima fase di conversione utilizzando il meccanismo standard descritto nella sezione 3.1.2

Capitolo 5

CONCLUSIONI

L'obiettivo di questa tesi è la realizzazione di un meccanismo che permetta di convertire documenti EARMARK [DPV11a][DPV11b] in formati XML. Come detto al capitolo 1, questo lavoro si inserisce all'interno di un progetto sviluppato dal dipartimento di Scienze dell'Informazione che tenta di mettere in relazione EARMARK, un meta-linguaggio che propone un approccio al markup basato sulle tecnologie del Semantic Web, con i linguaggi di markup tradizionali.

Il principale problema che abbiamo affrontato è quello dell'overlapping markup, che rappresenta il principale limite che deve essere risolto dai linguaggi di tipo embedded. Come detto al capitolo 2, XML e tutti i linguaggi di markup derivati da SGML impongono di organizzare i documenti secondo il modello ad albero che vincola gli elementi ad annidarsi secondo una rigidida organizzazione gerarchica. Questo schema non permette sempre di rappresentare in maniera esplicita gerarchie in overlap sullo stesso contenuto documentale, per cui l'unico modo di esprimere questi casi è di utilizzare dei trucchi sintattici che nascondano le strutture sovrapposte in un'altra forma (elementi che indicano i limiti di un altro elemento, elementi divisi in frammenti, riferimenti indiretti, ecc.). La necessità di rispettare la

gerarchia principale induce quindi ed offuscare l'importanza delle strutture secondarie, e di conseguenza anche la gestione e l'elaborazione diventa più complicata, impedendo di sfruttare le potenzialità degli strumenti forniti da XML.

Come abbiamo visto al capitolo 3, la soluzione fornita da EARMARK propone invece un diverso approccio al markup e alla gestione del problema dell'overlap: oltre a fornire un modello che permette di gestire le caratteristiche di documenti basati su alberi, EARMARK consente di esprimere in maniera esplicita strutture in overlap senza la necessità di utilizzare specifiche tecniche di codifica, rendendole trasparenti e facilmente comprensibili sia agli utenti che alle applicazioni. Un vantaggio evidente di EARMARK è la possibilità di accedere e navigare i documenti utilizzando strumenti molto diffusi e ampiamente supportati legati al Semantic Web. I documenti EARMARK sono infatti modellati attraverso ontologie OWL [HKP09], e le asserzioni che costituiscono i documenti sono semplici asserzioni RDF [MM04]. Di conseguenza è possibile utilizzare linguaggi di query come SPARQL [PS08] e un vasto insieme di strumenti esistenti per gestire strutture in overlap anche molto complicate: operazioni di ricerca e manipolazione di strutture arbitrariamente complesse che sono molto difficili o addirittura impossibili con le tecnologie legate ad XML risultano semplici ed immediate all'interno del modello di EARMARK.

Il problema principale da risolvere per raggiungere l'obiettivo di questa tesi è stato quello di mettere in relazione queste due tecnologie, studiando un meccanismo che permettesse di colmare il gap espressivo di questi linguaggi di markup. La soluzione descritta al capitolo 4 si chiama FRETTA (From EARMARK To Tag) e permette di convertire documenti EARMARK nel formato TEI, utilizzando un insieme di trucchi sintattici per convertire le strutture in overlap non direttamente esprimibili in XML. FRETTA propone di scomporre il problema in quattro fasi, che permettono di affrontare gli aspetti sintattici e semantici legati ai formati XML oggetto della conversione in maniera separata.

Durante i primi due passi della conversione viene effettuata una modifica alla struttura del documento in modo che rispetti la sintassi del formato di arrivo: durante il primo passo l'utente specifica i trucchi sintattici da utilizzare per esprimere le situazioni di overlap, mentre nella seconda fase FRETTA effettua un'analisi del documento e ne modifica la struttura secondo i trucchi specificati al passo precedente.

Nella terza fase FRETTA si occupa di interpretare secondo una certa semantica relativa a un certo formato specifiche parti del documento EARMARK, e quindi eseguire le modifiche definite per quella particolare interpretazione: se in questo passo l'utente specifica di utilizzare la semantica relativa ad un particolare formato (al momento è stata implementata la semantica del formato TEI P5 [SB05]), FRETTA utilizzerà tali semantiche per interpretare specifiche parti del documento EARMARK (sia elementi strutturali del documento che metadati ad essi relativi), e quindi eseguire le modifiche che l'utente ha indicato per quella particolare interpretazione (ad esempio iniettando alcuni di questi metadati all'interno della struttura del documento in specifici attributi ed elementi di markup).

L'ultimo passo si occupa infine di linearizzare il documento nel formato di arrivo. Il documento in input può essere quello generato nel passo II (EARMARK linearizzabile) o nel passo III (EARMARK semantico): l'utente ha la possibilità di eseguire solo la conversione strutturale, che consente di convertire le parti di documento non esprimibili in XML in strutture linearizzabili, oppure svolgere un ulteriore passo che interpreti e converta parti di documento secondo la semantica di uno specifico formato.

Il meccanismo di conversione realizzato da FRETTA rappresenta un primo passo nel progetto di realizzare un framework che consenta di mettere in relazione i diversi linguaggi di markup. In particolare, la scelta di utilizzare EARMARK come formato intermedio cui ricondursi durante la conversione permette di sfruttare i vantaggi introdotti da questo nuovo approccio al meta-markup: il modello di EARMARK permette infatti di esprimere in maniera esplicita caratteristiche documentali anche molto complesse che sono difficili da gestire con i linguaggi tradizionali.

La soluzione realizzata si occupa principalmente di gestire la complessità legata alla conversione di strutture in overlap: FRETTA è uno strumento completo che permette di risolvere queste situazioni implementando le principali tecniche di codifica sviluppate nel contesto XML. Inoltre il meccanismo di conversione realizzato prevede specifici meccanismi di estensione che permettono di potere aggiungere in futuro la gestione di ulteriori trucchi sintattici e tecniche di codifica, consentendo così di applicare FRETTA a contesti applicativi molto differenti fra loro.

Un possibile sviluppo di FRETTA è legato all'implementazione di specifiche

estensioni che permettano di effettuare una conversione fra formati di change tracking *OASIS Open Document* [WB11] utilizzato dalla suite *OpenOffice.org*, e *Office Open XML* [ISO08], sviluppato da *Microsoft*.

Bibliografia

[AB08] B. Adida, M. Birbeck. 2008. RDFa Primer, Bridging the Human and Data Webs. W3C Working Group Note 14 October 2008. http://www.w3.org/TR/xhtml-rdfa-primer/

[AH08] D. Allemagn, J. Hendler. 2008. Semantic Web for the Working ontologist, Effective Modeling in RDFS and OWL. Ed. Morgan Kaufmann, 2008.

[ALL07] J. Allsopp. 2007. Microformats: Empowering Your Markup for Web 2.0. New York, USA: Friends of ED Press, Berkeley, CA, 2007.

[AM08] M. d'Aquin, E. Motta, M. Sabou, S. Angeletou, L. Gridinoc, Lopez, D. Guidi. 2008. Toward a New Generation of Semantic Web Applications. Journal IEEE Intelligent Systems archive, Volume 23 Issue 3, May 2008. IEEE Educational Activities Department Piscataway, NJ, USA.

[BBC10] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon. 2010. XML Path Language (XPath) 2.0 (Second Edition). W3C Recommendation 14 December 2010. http://www.w3.org/TR/xpath20/

[BBG95] D. Barnard, L. Burnard, J. Gaspart, L. A. Price, C. M. Sperberg-McQueen, G. B. Varile. 1995. Hierarchical encoding of text: technical problems and SGML solutions. Computers and the Humanities 29: 211–231.

[BEC02] K. Beck. 2008. Test-Driven Development by Example. Ed. Addison Wesley. November 2002

[BG04] D. Brickley, R. V. Guha. 2004. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004.

http://www.w3.org/TR/rdf-schema/

[BHK88] D. Barnard, R. Hayter, M. Karababa, G. Logan, J. McFadden. 1988. SGML-based markup for literary texts: two problems and some solutions. Computers and the Humanities 22: 265–276.160.

[BNB10] F. Baader, D. Nardi, R. J. Brachman et al. 2010. The Description Logic Handbook. Cambridge University Press. May 2010. cap. 1,2,10.

[BPS08] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008.
http://www.w3.org/TR/xml/

[BT10] D. Becket, T. Berners-Lee. 2011. Turtle - Terse RDF Triple Language. W3C Team Submission 28 March 201.
http://www.w3.org/TeamSubmission/turtle/

[CKW08] P. Ciccarese, Wu, E., Kinoshita, J., Wong, G., Ocana, M., Ruttenberg, A., Clark, T. 2008. The SWAN Biomedical Discourse Ontology. In Journal of Biomedical Informatics, v.41(5), pp. 739-751. Elsevier.

[CM95] S. Cyrano de Bergerac, D. Moncond'huy. La mort d'Agrippine. Tableronde, November 1995.

[CRD87] J. H. Coombs, A. H. Renear, S. J. DeRose. 1987. Markup systems and the future of scholarly text processing http://vision.unipv.it/stm-cim/articoli/p933-coobs.pdf

[DDG02] S. DeRose, R. Daniel Jr., P. Grosso, E. Maler, J. Marsh, N. Walsh. 2002. XML Pointer Language (Xpointer). W3C Working Draft 16 August 2002.
http://www.w3.org/TR/xptr/

[DDM90] S. J. DeRose, D. G. Durand, E. Mylonas, A. H. Renear. "What is Text, Really?" Journal of Computing in Higher Education 1(2): 3-26.

[DER04] S. DeRose. 2004. Markup overlap: A review and a horse. In the

Proceedings of Extreme Markup Languages 2004. Montreal, Canada.

[DO02] P. Durusau, 2002. M. B. O'Donnell. Coming down from the trees: Next step in the evolution of markup In Extreme Markup Languages, 2002.

[DPV10] A. Di Iorio, S. Peroni, F. Vitali. 2010. Handling markup overlaps using OWL. In P. Cimiano, & H. S. Pinto (Eds.), Proceedings of the 17th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2010) (pp. 391-400). Heidelberg, Germany: Springer

[DPV11a] A. Di Iorio, S. Peroni, F. Vitali. 2011. Using Semantic Web technologies for analysis and validation of structural markup. To be published in International Journal of Web Engineering and Technologies. Inderscience Publisher.

[DPV11b] A. Di Iorio, S. Peroni, F. Vitali. 2011. A Semantic Web Approach To Everyday Overlapping Markup. To be published in Journal of the American Society for Information Science and Technology. Wiley.

[DSV11] A. Di Iorio, S. Peroni, F. Vitali. 2011. Using Semantic Web technologies for analysis and validation of structural markup. Submitted for publication in International Journal of Web Engineering and Technologies.

[DUR02] P. Durusau. 2002. Visualizing overlapping hierarchies in textual markup. In Joint international conference ALLC/ACH, 2002.

[GOL81] C. Goldfarb. 1981. A Generalized Approach to Document Markup. In Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation, New York: ACM.

[HFB09] J. Hebeler, M. Fisher, R. Blace, A. Perez-Lopez. 2009. Semantic Web Programming. Wiley Publishing, Indianapolis, Indiana.

[HKP09] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, S. Rudolph. 2009. OWL 2 Web Ontology Language Primer. W3C Recommendation

27 October 2009.
http://www.w3.org/TR/owl2-primer/

[HPB04] I. Horrocks, P. F. Patel-Schneider, H. Boley, s. Tabet, B. Grosof, M. Dean. 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004.
http://www.w3.org/Submission/SWRL/.

[HS01] C. Huitfeldt, C. M. Sperberg-McQueen. 2001. TexMECS: An experimental markup meta-language for complex documents. Working paper of the project Markup Languages for Complex Documents (MLCD), University of Bergen.

[ISO08] Technical Committee JTC 1/SC 34. 2008. ISO/IEC 29500-1: Information technology -- Document description and processing languages -- Office Open XML File Formats -- Part 1: Fundamentals and Markup Language Reference.

[JLS04] H. V. Jagadish, Laks V. S. Lakshmanan, Monica Scannapieco, Divesh Srivastava, and Nuwee Wiwatwattana. 2004. Colorful XML: One hierarchy isn't enough. Proceedings of the 2004 ACM SIGMOD International, pag. 251–262.
http://www.cs.uiuc.edu/class/fa05/cs591han/sigmodpods04/sigmod/pdf/R-456.pdf

[MM04] M. Manola, E. Miller. 2004. RDF Primer. W3C Recommendation.
http://www.w3.org/TR/rdf-primer/.

[MUR95] M. Murata. 1995. File format for documents containing both logical structures and layout structures. In Electronic publishing 8, 295–317.

[MVZ08] P. Marinelli, F. Vitali, S. Zacchiroli. 2008. Towards the unification of formats for overlapping markup. In New Review of Hypermedia and Multimedia, v. 14(1), pp. 57-94. Taylor and Francis, ISSN 1361-4568.

[NEL80] T. Nelson, 1980. Literary Machines: The report on, and of, Project

Xanadu concerning word processing, electronic publishing, hypertext, thinkertoys, tomorrow's intellectual... including knowledge, education and freedom – Mindful Press, Sausalito, CA, USA.

[PS08] E. Prud'hommeaux, A. Seaborne. 2008. SPARQL Query Language for RDF. W3C Recommendation 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/

[REI80] B. Reid. 1980. A High-Level Approach to Computer Document Formatting. In Proceedings of the 7th Annual ACM Symposium on Programming Languages, New York: ACM.

[RMD96] A. Renear, E. Mylonas, D. Durand. 1996. Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies. In Hocky, S. and Ide, N. (eds), Research in Humanities Computing. Oxford University Press, Oxford, UK. http://www.stg.brown.edu/resources/stg/monographs/ohco.html

[SB05] C. M. Sperberg-McQueen, L. Burnard. 2005. TEI P5 Guidelines for Electronic Text Encoding and Interchange (revised). The Association for Computers and the Humanities.

[SB90] C. M. Sperberg-McQueen, L. Burnard. 1994. A Gentle Introduction to SGML. Chapter 2 of Guidelines for Electronic Text Encoding and Interchange. (TEI P3), Text Encoding Initiative. Chicago, Oxford 1990, 1992, 1993, 1994. http://www-sul.stanford.edu/tools/tutorials/html2.0/gentle.html

[SC09] D. Schmidt, R. Colomb, 2009. A data structure for representing multi-version texts online. International Journal of Human-Computer Studies.

[SH00] C. M. Sperberg-McQueen, C. Huitfeldt. 2000. GODDAG: A data structure for overlapping hierarchies. In DDEP/PODDP, pages 139–160, 2000. http://cmsmcq.com/2000/poddp2000.html

[SH08] C. M. Sperberg-McQueen, C. Huitfeldt. 2008. Markup Discontinued: Discontinuity in TexMecs, Goddag structures, and rabbit/duck grammars. In Proceedings of Balisage: The Markup Conference, August 12-15, 2008.
http://www.balisage.net/Proceedings/vol1/html/
Sperberg-McQueen01/BalisageVol1-Sperberg-McQueen01.html

[SHR00] C. M. Sperberg-McQueen, C. Huitfeldt, A. Renear. 2000. Meaning and Interpretation of Markup. Paper originally presented at ALLC/ACH 2000, Glasgow, and at Extreme Markup Languages 2000, Montreal.

[SW06] O. Schonefeld, A.Witt. 2006. Towards validation of concurrent markup. In Proceedings of the Extreme Markup 2006, Montréal, Canada
http://conferences.idealliance.org/extreme/html/2006/Schonefeld01/EML2006Schonefeld01.html

[TEN08] J. Tennison. 2008. Representing Overlap in XML. Article from “Jeni's Musings”
blog, http://www.jenitennison.com/blog/node/97.

[TP02] J. Tennison, W. Piez. 2002. The Layered Markup and Annotation Language (LMNL). Paper presented at the Late breaking at Extreme Markup. Montreal, Canada.

[WB11] R. Weir, M. Brauer. 2011. OASIS Open Document Format for Office Applications v1.2 (OpenDocument). OASIS Standard. 26 March 2011.
http://docs.oasis-open.org/office/v1.2/cs01/OpenDocument-v1.2-cs01.pdf

[WIT04] A. Witt. 2004. Multiple hierarchies: new aspects of an old solution. Proceedings of Extreme Markup Languages. Montréal, Québec. August 2-6, 2004. pag. 55-85.

alla mia famiglia ed Eugenia