OntoReST: A RST-based Ontology for Enhancing Documents Content Quality in Collaborative Writing

HALA NAJA-JAZZAR [1]
NISHADI DE SILVA [2]
HALA SKAF-MOLLI [3]
CHARBEL RAHHAL [3]
PASCAL MOLLI [3]

[1]LaMA Laboratory - Lebanese University, Tripoli, Lebanon
hjazzar@ul.edu.lb, hala.naja@hotmail.com
[2] School of Electronics and Computer Science, University of Southampton, UK
n.desilva@ecs.soton.ac.uk
[3] LORIA – INRIA Lorraine, Université de Nancy, France
skaf@loria.fr, Charbel.Rahal@loria.fr, molli@loria.fr

Abstract. Collaborative writing is the process by which more than one author contributes to the content of a document. Although, multi-synchronous collaboration is very efficient in reducing task completion time, it is well known for producing documents of poor-quality content. Most existing collaborative writing environments do not really check the logical arrangement of documents portions (i.e. sentences, paragraphs,...). They rely on authors to verify the content quality of the document. This imposes a severe overhead on the authors to achieve efficient collaboration. To address this issue, we use semantic web technologies and a discourse theory called Rhetorical Structure Theory (RST) with the aim to reduce the overhead of consistency checking. We develop *OntoReST*, an ontology based on RST that helps detect incoherent texts automatically. *OntoReST* also provides authors with valuable information about the semantic structure of texts which contributes towards enhancing documents content quality.

Keywords: Collaborative writing, Ontology, Document content quality, Semantic web.

(Received April 14, 2009 / Accepted June 07, 2009)

1 Introduction

Collaborative writing is the process by which more than one author, in addition to sharing opinions, contributes to the content of a document [7]. Collaborative writing is standard practice in technical and scientific settings; some examples include research papers, software development, proposals for funding and user manuals. When collaboration is efficiently managed, the advantages of working in a group include improved efficiency, reduced errors, and increased the benefits of different viewpoints and expertise [10]. If collaboration is poorly supported, it will lead to inconsistencies, misunderstandings, conflicts, redundant work and coordination problems. The nature of collaboration varies extensively in terms of the group writing strategies, proximity and synchronicity of group activities. For instance, collaborative writing can be done in parallel synchronously, asynchronously or multi-synchronously [9, 13]. Synchronous work, with a joint-writing strategy, can give good results but is limited to small groups working for a short period of time. Asynchronous work with turn-taking strategies, allows to work distributed in time but does not allow task parallelization. The multi-synchronous interaction mode allows people to work in parallel synchronously or asynchronously while being distributed in time and space. It is the least restrictive collaborative writing strategy.

This working mode is well known in software engineering. Software engineers commonly use version control systems or distributed version control system to achieve high parallelization of tasks and reduce development time. This high level of concurrency is potentially risky and can lead to software inconsistencies. Fortunately, software engineers can compile software and run automatic tests in continuous integration strategy [3] to limit the risk of inconsistencies.

Unfortunately, we cannot reuse these proven efficient collaborative strategies outside the software engineering world. Currently, we lack the presence of automatic testing mechanisms for checking document content quality according to some specifications. Text documents do not have typed objects to reason about, and therefore cannot be “compiled” in order to verify some type safety violation.

Multi-synchronous collaboration mode greatly increases the risk of misaligned contributions by individual authors. Despite each section may be well constructed, all sections may not ‘fit’ logically when placed together. While this is easy to correct in short texts, the problem is much harder in large, multi-authored documents. This is what we refer to as **content quality** which is the focus of this paper.

Content quality is poorly supported by existing Multi-synchronous collaborative writing environments. In these environments, each author works on her own copy of the shared data. The system is correct if: (1) it eventually converges to an idle state where all copies are identical (2) user intentions are preserved [15]. ’Intention’ means that if an operation produced an effect when generated, this effect must be observed in the same way by all users. Collaborative environments help authors by providing awareness about concurrent changes. Next, authors have to verify that each local operation is compatible with all other concurrent operations. We want to leverage this stage by providing more information about the context and impact of modifications. This requires us to define more clearly what is meant by “content quality”.

What we mean by content quality is the ease with which a document can be read and understood. While multiple factors such as grammar and punctuation can affect this, **the logical progression** of the ideas presented is perhaps the one with the most impact.

In this paper, the words incoherence and inconsistency are used interchangeably. They designate missarrangements in documents portions (i.e. sentences, paragraphs,...) that negatively affect the general meaning of the document.

In this paper, we use techniques from the semantic web domain to address the problem of poor-quality content documents during collaborative writing. More precisely, we define an ontology called *OntoReST* based on the theory on Rhetorical Structure Theory (RST) defined by Mann and Thompson [8]. *OntoReST* turns the document contents into a machine readable and structured form which allows the detection of illogical arrangements of document portions. Such ontologies can be used to turn text into typed objects. Consequently, multi-synchronous collaboration interaction mode can be used in order to achieve more efficient collaboration for producing text documents.

So, first we give a brief description of RST and show how the structure of a text can be analysed using it. In section 3, we define *OntoReST* formally using OWL and description logic DL. Next, in section 4, we detail the required steps to manipulate this ontology during collaborative writing and describe the applied merging algorithm using an example. We also discuss related work in the field and finally, present our conclusions and directions for future work.

2 Rhetorical Structure Theory (RST)

There are several discourse theories developed by linguists to analyse the structure of texts. We have chosen Rhetorical Structure Theory (RST) [8] for its simplicity, clear relationship definitions and its ability to render itself into formal descriptions. RST attributes the coherence of a text to implicit logical relationships such as ’Motivation’, ’Background’ and ’Elaboration’ that exist between portions of the text. The rest of this section gives a brief overview of how a RST analysis can be done and highlights parts of the process essential to the discussions in this paper.

2.1 Analysing a Text Using RST

The first step in a RST analysis is to divide the text into non-overlapping, functionally independent segments[8]. As an example, we use the text below to demonstrate the segmentation.

[**Text 1:**] [*1:The problem with existing writing software is their inability to detect semantic problems in documents.*] [*2:OntoReST is the result of combining the techniques behind onotologies and those related to RST.*] [*3:When combined with existing writing tools, it can help improve the quality of documents by alerting authors to possible semantic inconsistencies.*]
During a bottom-up analysis, the second step is to identify logical relationships that exist between pairs of segments. For instance, in the above example, we see segment 3 to be providing motivating information to the

statement in segment 2 (*i.e.* Motivation relationship).

Segments in a relationship can play one of two roles: a **nucleus** or a **satellite**. A nucleus is considered to be an important segment, essential to the understanding of the text. A satellite is not as critical but does provide supporting material.

More information about RST can be found in Mann and Thompson's paper [8] where 23 relationships are defined. Henderson and De Silva [5], however, considered 23 to be too many for technical writing and began selecting a subset of relationships that were sufficient for analysing technical documents. In [2], a user study has shown that technical authors found a set of 9 relationships adequate for their analysis.

In the analysis, segments involved in a relationship collectively form a **span**. A span can in turn become part of another relationship. For instance, in our example, the span of segments 2 and 3 is identified as being in a BACKGROUND relationship with segment 1 (*i.e.* segment 1 provides background information that helps understand the significance of BOTH segments 2 and 3). Hence, the analysis is a recursive process and continues until all the segments are assembled into a tree of relationships. This is called a **RSTree**.

2.2 RSTrees Properties

The important point about RSTrees is that Mann and Thompson conjecture that producing a well-formed RSTree for a text indicates that a text is coherent. This is a useful measure in our work where we apply RST to detect incoherent texts. They define four properties that determine if a RSTree is well formed. They are:

1. ***Completedness:*** One schema application (the root) should cover the entire text.

2. ***Connectedness:*** Each text span/segment, apart from the span that covers the entire text, should be a minimal unit in the tree or part of another schema application.

3. ***Uniqueness:*** Each text span/segment should have only one parent (*i.e.* each schema application consists of a different set of text spans/segments).

4. ***Adjacency:*** Only adjacent text spans/segments can be grouped together to form larger spans.

We make use of these properties in *OntoReST* to evaluate the quality of documents written collaboratively.

3 Ontologies for documents description

An ontology describes basic concepts in a domain and defines relations among them. It is composed of concepts, properties, relations and restrictions on properties. We formalize the RST theory as an ontology. This allows to take advantages of the semantic web by turning the content of document into a machine readable and structured form. It provides also a common knowledge base for the authors. Moreover, it is possible to detect automatically semantic problems and to make interesting queries on the document. For example, by selecting all the **Nucleus** we can produce a summary of the document.

We have defined three ontologies (see figure 1):

- **Document structure ontology:** captures the internal structure of the document (sections, sentences, etc).

- **Rhetorical ontology*OntoReST***: models the document in terms of its rhetorical elements (*i.e.* segments, spans and RST relationships). This allows the detection of semantic inconsistencies in documents.

- **Annotation ontology**: annotates the sentences and the sections of the document. It also captures additional meta data about the document. This is helpful in classifying the documents according to their types, authors and topics.

The definition of three separate ontologies allows flexible modification of the ontologies and is inspired from [16]. In this paper, our major interest is detecting inconsistency in the documents. For this reason, we will focus just on the rhetorical ontology *OntoReST*.

3.1 Rhetorical Ontology *OntoReST*

The rhetorical ontology captures the semantics of the text by using RST. It models the segments, spans and rhetorical relations. It also uses the four properties for well-formed RSTrees to detect incoherences in the document. We only use the subset of 9 relations identified in [5, 2].

We identify five main concepts: *Document, Element, Span, Segment and RhetoricalRelation* as shown in figure 2. In this figure, we omit inverse properties for simplicity.

A *Document* is composed of an ordered sequence of segments and has two sets of spans and rhetorical relations. It has also the following properties:

- *hasID*: a unique identifier given by the system for the document.

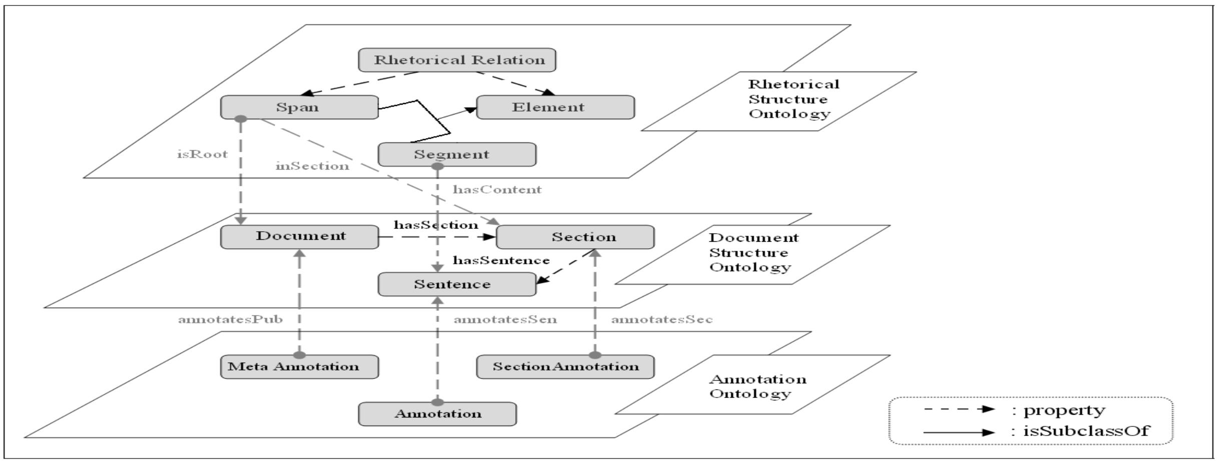

Figure 1: Ontology Layers

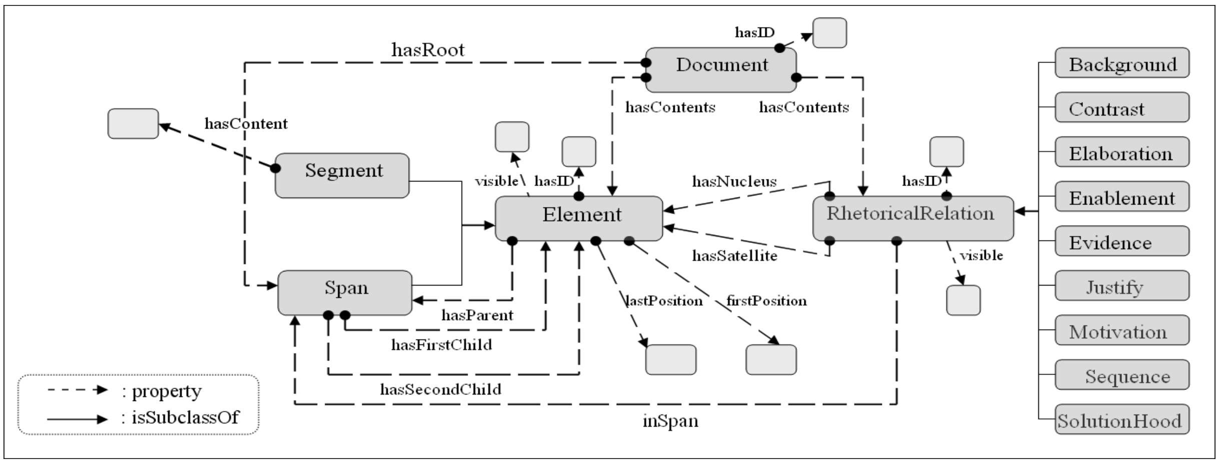

Figure 2: The Rhetorical Ontology *OntoReST*

- *hasContents*: links a document to its segments, spans and rhetorical relations. The textual content of a document is the *hasContent* values of its all ordered segments. This property is the inverse property of *containingDoc*.

- *hasRoot*: indicates the root of a document, which is the span that covers the entire text. Its inverse property is *isRoot*.

An *Element* can be either a Segment or a Span. It is used to avoid repetition of common properties in Segments and Spans. It has the following properties:

- *hasID*: is a unique identifier given by the system for each element.

- *firstPosition* and *lastPosition*: are the position of the segment in the document. They have equal value for a segment. For a span, they indicate the position of the first and last segments covered by the span in the document. We use them for the adjacency property.

- *visible*: is the status of the element. Its range is boolean and has “true" as default value. It turns to “false" when the element is deleted. We add this property because we do not delete physically the element, but rather we mark it as invisible. This property will be set by the merging algorithm as we will see later.

- *hasParent*: indicates the parent of an element. Each element in the document has a parent property except the root.

- *containingDoc*: indicates the document containing the elements.

A *Segment* is a sentence. It inherits the properties of an element and has an additional property *hasContent*.

A **Span** covers two adjacent segments and is always linked to one rhetorical relation. Span inherits the properties of an element and has the additional following properties:

- *hasFirstChild* and *hasSecondChild*: are the first and the second child elements of a Span.
- *isRoot*: indicates the parent document of the root span. Its inverse property is *hasRoot*.
- *hasRstRelation*: indicates the rhetorical relation existing between the children of the span. The children of a span are the nucleus and satellite of that relation.
- *changed*: is the status of the span. It is “false" by default and set to “true", after deleting one of its children or the *hasRstRelation* property.

The *changed* property allows to propagate modifications to the concerned spans and relations in the RSTree, and provides some awareness that helps the authors to localize the modified parts of the documents.

A *Rhetorical Relation* is the rhetorical relation that holds between two elements. It is always linked to a span. It has the following properties:

- *hasID*: is a unique identifier given by the system for each relation.
- *hasName*: is the name of the relation such as *Motivation* or *Elaboration*. Its domain is a Relation and its range is a list of relations' names. According to the RST theory, the name of the relation specifies the order of the nucleus and the satellite *i.e.* nucleus is before satellite or the opposite.
- *hasNucleus* and *hasSatellite*: represent the nucleus and satellite of the relation respectively. Their values are the first and the last child of the linked span *i.e.* they refer to the first and last child properties of the related span.
- *visible*: is the status of the relation. Its range is boolean and has “true" as default value. It turns to “false" when the relation is deleted.
- *inSpan*: this property is the inverse property of the *hasRstRelation*.

3.2 Formal Specification of *OntoReST*

We use both OWL (Web Ontology Language) and Description Logic to formalize *OntoReST*. We write the class axioms, the property axioms, and the constraints in OWL DL which is the most investigated species of OWL [1]. OWL DL has different syntaxes. However, the normative syntax of OWL DL is the abstract syntax. OWL can be seen as an alternate notation for Description Logic Language $\mathcal{SHOIN}(\mathtt{D})$. Table 1 presents the concepts of the *OntoReST*.

Table 2 presents the object and data properties. In this table, we skipped some identical properties for simplicity. Both tables represent a mapping between the OWL DL abstract syntax and the syntax of the Description Logic $\mathcal{SHOIN}(\mathtt{D})$.

4 *OntoReST* in Collaborative Writing

Using *OntoReST* to maintain consistency during collaborative writing requires the following steps:

- *Ontology instantiations*: Each instance of concept is created locally at a user's site. First, when the author adds a sentence, the system will detect this modification as an operation and creates an instance of the segment concept. Second, the operation is sent and integrated at all other users' sites. Finally, if there are no modifications, the replicated instances will be the same at all sites. In section 4.1, we define operations to instantiate this ontology.
- *Merging algorithms*: To integrate remote modifications, we use the *Tombstone Transformation Functions* algorithm [11] (TTF) as mentioned in section 4.3.
- *Inconsistency checker*: There is a significant difference between using RST in collaborative writing and traditional applications of RST. Normally, RST is applied to a ‘static’ text. However, in collaborative writing, the text varies too often and its corresponding RSTree changes too. We have to ensure that the new RSTree respects the four properties defined in the RST theory, as detailed in section 4.4.

4.1 Populating Ontology

In this section, we describe the process of ontology's instantiation during the edition.

During the edition, the changes made by the authors are detected by the system as follows:

OWL Abstract syntax	DL syntax
Class axioms	
SubClassOf(Document Thing)	Document $\sqsubseteq \top$
SubClassOf(Element Thing)	Element $\sqsubseteq \top$
SubClassOf(Span Element)	Span $\sqsubseteq$ Element
SubClassOf(Segment Element)	Segment $\sqsubseteq$ Element
EquivalentClasses(Element unionOf(Span Segment))	Element $\equiv$ Span $\sqcup$ Segment
DisjointClasses(Span Segment)	Span $\sqcap$ Segment $\sqsubseteq \bot$
EquivalentClasses(RheRelation)	RheRelation $\equiv$
unionOf(Background ... SolutionHood))	Background $\sqcup$ ... $\sqcup$ SolutionHood
DisjointClasses(Background, Contrast)	Background $\sqcap$ Contrast $\sqsubseteq \bot$, ...
DisjointClasses(Sequence, SolutionHood)	Sequence $\sqcap$ SolutionHood $\sqsubseteq \bot$

Table 1: The Rhetorical concepts in OWL and DL

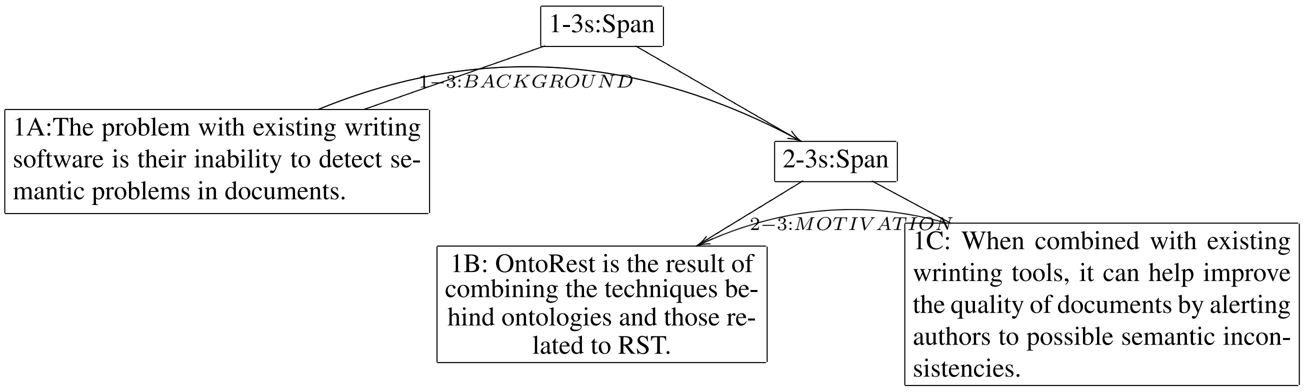
1-3s:Span
1−3:BACKGROUND
1A:The problem with existing writing software is their inability to detect semantic problems in documents.
2-3s:Span
2−3:MOTIVATION
1B: OntoRest is the result of combining the techniques behind ontologies and those related to RST.
1C: When combined with existing wrinting tools, it can help improve the quality of documents by alerting authors to possible semantic inconsistencies.

Figure 3: RSTree of *AnnotatedText1*

- *addSeg(position, hasID, content, sid)* adds an instance of segment with the specified position and the text content. *sid* is the identifier of the site generating the operation. The *sid* is necessary for the merging algorithm.

- *delSeg(position)* deletes logically a segment at the given position. The *visible* property of the segment is set to false. There is no physical deletion of segments to ensure the convergence [11]. If the segment has a parent span, the *changed* property of its parent will be set to true.

- *addSpan(hasID, hasFirstChildID, hasSecondChildID)* creates an instance of span with the required properties.

- *delSpan(hasID)* deletes logically a span. The *visible* property of the span is set to false. If the span has a parent span, then the *changed* property of its parent is set to true.

- *addRel(hasID, NucleusID, SatelliteID, SpanID, hasName)* adds a rhetorical relation between the children of span *SpanId*. *NucleusId*, respectively *SatelliteId*, is the *spanId* child which is nucleus, respectively satellite of the created relationship.

- *delRel(hasID)* deletes logically a relation. The *visible* property of the relation is set to false. The *changed* property of the span linked to this relation is set to true.

We define only *add* and *delete* operations. Because, during the merge the *update* operation is detected as *delete* followed by *add* by the diff algorithms.

Let us consider a scenario where an author is working on $site1$. She wants to write the *Text 1* of section 2 but this time with rhetorical annotations.

She modifies her local copy through generating operations. The system will detect these changes as the following sequence of operations:

```
S =[
addSeg(1,1A,"The problem ...",1);
addSeg(2,1B,"OntoReST is...", 1);
addSeg(3,1C,"When combined ...",1);
addSpan(2-3s,1B,1C );
addRel(2-3,1C,1B,2-3s,"Motivation");
```

OWL Abstract syntax	DL syntax
Property axioms	
ObjectProperty(hasFirstChild domain(Span) range(Element))	$\top \sqsubseteq \forall$ hasFirstChild$^-$.Span $\top \sqsubseteq \forall$ hasFirstChild.Element
restriction(hasFirstChild maxCardinality(1) minCardinality(1))	Span $\sqsubseteq$ (= 1 hasFirstChild)
ObjectProperty(hasNucleus domain(RheRelation) range(Element))	$\top \sqsubseteq \forall$ hasNucleus$^-$.RheRelation $\top \sqsubseteq \forall$ hasNucleus.Element
restriction(hasNucleus minCardinality(1) maxCardinality(2))	RheRelation $\sqsubseteq$ ($\geq$1 hasNucleus) $\sqcap$ ($\leq$2 hasNucleus)
ObjectProperty(hasParent domain(Element) range(Span))	$\top \sqsubseteq \forall$ hasParent$^-$.Element $\top \sqsubseteq \forall$ hasParent.Span
ObjectProperty(inSpan domain(RheRelation) range(Span) inverseOf(hasRstRelation))	$\top \sqsubseteq \forall$ inSpan$^-$.RheRelation $\top \sqsubseteq \forall$ inSpan.Span inSpan $\equiv$ hasRstRelation$^-$
DatatypeProperty(firstPosition domain(Element) range(String))	Element $\sqsubseteq \exists$ firstPosition.String
DatatypeProperty(hasID domain(unionOf(Element RheRelation Document)) range(Integer))	$\top \sqsubseteq \forall$ hasID$^-$.(Element $\sqcup$ RheRelation) Element $\sqcup$ RheRelation $\sqcup$ Document $\sqsubseteq \exists$ hasID.Integer
DatatypeProperty(visible range(Boolean))	Element $\sqcup$ RheRelation $\sqsubseteq \exists$ visible.Boolean

Table 2: The Rhetorical properties in OWL and DL

```
addSpan(1-3s,1A,2-3s);
addRel(1-3,1A,2-3s,1-3s,"Background")]
```

The system will build the RSTree for *Annotated-Text1* as depicted in figure 3.

4.2 Concurrent Writing

Now consider that two authors A and B, working on $site2$ and $site3$ respectively, are writing the *Annotated-Text1* of figure 3. Each author has his own copy of the text.

Author A decides to delete the segment in position 2. In order to preserve local consistency of the RSTree, the system will propagate this deletion to its parent span and associated relation. Span $2-3s$ and relation $2-3$ are logically deleted. Moreover, the parent of span of $2-3s$ will be replaced by $1-3's$ and relation $1-3$ by $1-3'$ as shown in figure 4.

The changes performed by $authorA$ will be detected by the system as the following sequence of operations $P1$:

```
P1= [delSeg(2); delSpan(2-3s);
delRel(2-3); delSpan(1-3s);
delRel(1-3);addSpan(1-3's,1A,1C);
addRel(1-3',1A,1C,"Background")]
```

At the same time, author B performs concurrent operations (see figure 5). She adds a new segment:“This is a work in progress." at position 4 and two relations.

The system produces the following set of operations $P2$:

```
P2=[addSeg(4,1D,"This...",3);
delSpan(1-3s); delRel(1-3);
addSpan(2-4s,2-3s,1D);
addRel(2-4,1D,2-3s,"Elaboration");
addSpan(1-4s,1A,2-4s);
addRel(1-4,1A,2-4s,"Background");]
```

4.3 Merging Ontological Data

In this section, we will detail through an example how we merge the above concurrent operations. Authors A and B have generated P_1 and P_2 respectively, so the copies hosted on $site2$ and $site3$ are diverging now *i.e.* they have different content. Both sites exchange their operations and run the integration process. In order to converge, the system has to ensure $Merge(P1, P2) = Merge(P2, P1)$. Unfortunately, this property would not be ensured by traditional merge algorithms.

This problem is well-know in CSCW community. The Operation Transformation (OT) framework [4] has been developed to ensure convergence in these conditions. In [12], we defined a set of all transformation functions dealing with concurrent operations, and ensure convergence of semantically annotated documents with RST.

As shown in figure 6, the final state is converged towards an inconsistent value which is the result of merg-

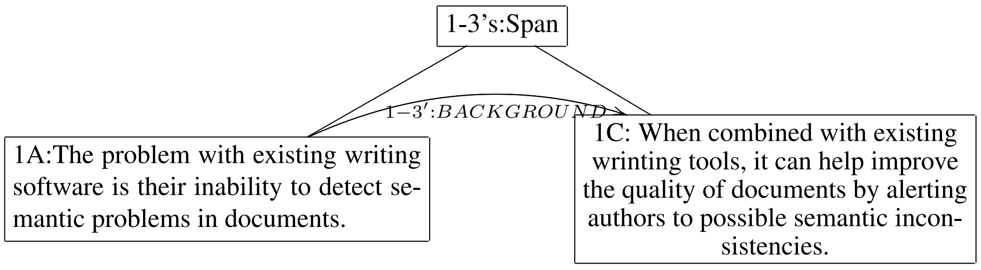
1-3's:Span
1−3′:BACKGROUND
1A:The problem with existing writing software is their inability to detect semantic problems in documents.
1C: When combined with existing wrinting tools, it can help improve the quality of documents by alerting authors to possible semantic inconsistencies.

Figure 4: *AnnotatedText1* for author A

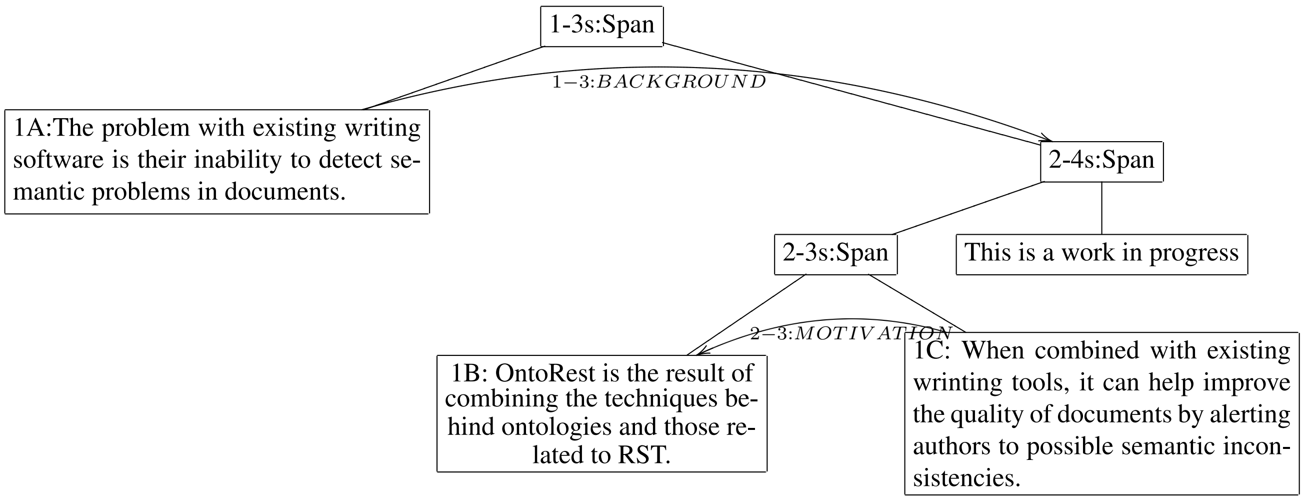
1-3s:Span
1−3:BACKGROUND
1A:The problem with existing writing software is their inability to detect semantic problems in documents.
2-4s:Span
2-3s:Span
This is a work in progress
2−3:MOTIVATION
1B: OntoRest is the result of combining the techniques behind ontologies and those related to RST.
1C: When combined with existing wrinting tools, it can help improve the quality of documents by alerting authors to possible semantic inconsistencies.

Figure 5: *AnnotatedText1* of author B

ing the two RSTrees. The resulting value does not respect the rhetorical properties. For example, the segment at position 1 violates the uniqueness property. The resulting **RSTree** will help the authors to better understand the reasons of the semantic inconsistency and to locate exactly the segments of the text which are responsible for this inconsistency. In the next section, we detail our *inconsistency checker*.

4.4 Inconsistency Checker

In our approach, we consider a document as coherent if it respects the four properties of the RST (see section 2). Therefore, we formalize these properties as constraints in the *OntoReST* ontology. These constraints are checked continuously. As result, individual author produces coherent documents and detects any inconsistency after merging concurrent modifications.

To check the inconsistent instances of the *OntoReST*, we formalize the violation of the coherence properties of the RST in DL on an instance d of the document $Document(d)$ as follows :

1. **Completedness Violation (CompV)** document d has a number of roots different from one.

   $\texttt{CompV} \equiv \neg\ (= 1\ \texttt{hasRoot})$

2. **Connectedness Violation (ConV)** For document *d*, an inconsistent element that violates the connectedness property is a visible element that has no parent and is not a root of *d*.

   $\texttt{ConV} \equiv \texttt{Element} \sqcap \exists\ \text{visible}.\{\text{true}\} \sqcap \exists\ \texttt{containingDoc}.\{\texttt{d}\} \sqcap \forall\ \text{hasParent}.\bot \sqcap \forall\ \text{isRoot}.\neg\{\text{d}\}$

3. **Uniqueness Violation (UniV)** For document *d*, an inconsistent element that violates the uniqueness property is a visible element that has a number of parents different than one.

   $\texttt{UIE} \equiv \texttt{Element} \sqcap \exists\ \text{visible}.\{\text{true}\} \sqcap \exists\ \text{containingDoc}.\{\text{d}\} \sqcap \neg\ (=1\ \texttt{hasParent})$

4. **Adjacency Violation (AdjV)** For a visible span in document *d*, there exists a visible element (segment or span) between the last position of its first child and the first position of its second child.

   $\texttt{C1} \equiv \texttt{Span} \sqcap \exists\ \text{visible}.\{\text{true}\} \sqcap \exists\ \text{containingDoc}.\{\text{d}\}$

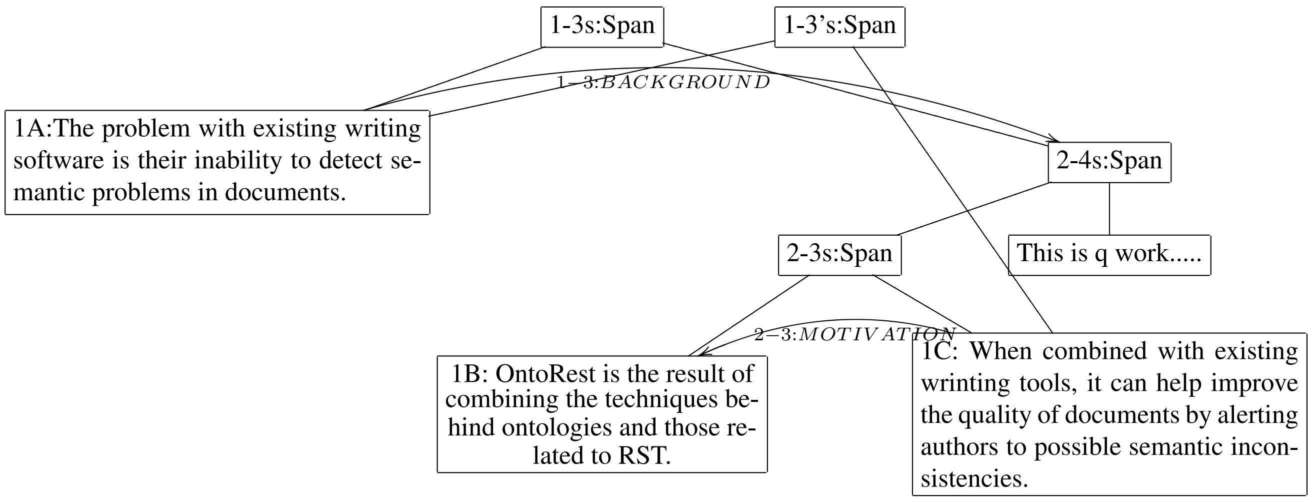
1-3s:Span
1-3's:Span
1−3:BACKGROUND
1A:The problem with existing writing software is their inability to detect semantic problems in documents.
2-4s:Span
2-3s:Span
This is q work.....
2−3:MOTIVATION
1B: OntoRest is the result of combining the techniques behind ontologies and those related to RST.
1C: When combined with existing wrinting tools, it can help improve the quality of documents by alerting authors to possible semantic inconsistencies.

Figure 6: Merging Result

`C2` $\equiv$ `Element` $\sqcap \exists$ visible.{true} $\sqcap \exists$ containingDoc.{d}

Let s and e be $C1(s)$ and $C2(e)$:

```
AdjV ≡
(> firstPosition.{e}
lastPosition.hasFirstChild.{s})
⊓
(< lastPosition.{e}
firstPosition.hasSecondChild.{s}).
```

We have implemented and verified our *OntoReST* and rhetorical constraints in Protégé and Protégé Axiom Language (PAL) respectively.

5 Related works

In the collaborative writing domain, most studies on semantic consistency are based on constraints. Some approaches allow the violation of the constraints. When violated, reparation is done either automatically, like in [14], or manually. Others prevent constraints violation like in [6]. If an operation violates the constraints, the operation is canceled. Both approaches in [14] and [6] bring about lost updates which is not ideal.

Constraints can be specific to an application and concern more the document structure. However, they cannot capture the co-author's understanding and logical reasoning about the text. RST provides this need by attributing relationships between its segments. These relationships create an overall effect on the reader, resulting in a better understanding of the text. Consequently, when authors exchange documents, they also pass on their knowledge via the attached RST relationships. In our work, we evaluate the semantic consistency of documents, by using the RST properties as our constraints.

The project SALT (Semantically Annotated Latex) has some common features with our work. In [16], the authors propose a framework for authoring and annotating LaTeX documents. They develop ontology based on RST. The authors add RST-based semantic tags to their LaTeX documents while editing. SALT does not consider collaborative work. In our work, the *OntoReST* ontology is used not only to add semantic annotations within the document but also to evaluate the document's level of coherence during collaborative writing. By constantly maintaining and checking the four properties, we are able to detect inconsistencies and alert the authors to such areas. We anticipate to integrate our work easily into the SALT framework so that our RST capability can be extended into LaTeX documents too.

6 Conclusions and Future Work

The problem we have tackled in this paper is the lack of support in current collaborative writing tools to detect semantic problems in texts. We have combined ideas from a discourse theory, called Rhetorical Structure Theory (RST), and ontologies to develop OntoReST. RST has provided us with a way to formally identify what it means for a text to be coherent (or semantically sound). RST attributes the coherence of a text to underlying relationships between its segments. These relationships create an overall effect on the reader which contributes towards a better understanding of texts. The creators of RST also define some properties which we have formalised and used in our work to evaluate if the text is semantically sound.

OntoReST offers a novel application of annotations, where the focus is to achieve more coherent documents in collaborative writing. It benefits from having the advantages of ontologies such as providing a common knowledge base for authors, and searching easily for data through queries. We make use of a merging technique to ensure the convergence of the OntoReST, *i.e.* getting identical RSTrees for documents on all the authors' sites leaving no room for confusion.

This combination of different strands of research (RST, ontologies and collaborative writing) is novel. As future work, we intend to evaluate its practical use and identify its shortcomings.

References

[1] de Bruijn, J., Lara, R., Polleres, A., and Fensel, D. Owl dl vs. owl flight: conceptual modeling and reasoning for the semantic web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2005. ACM.

[2] De-Silva, N. *A narrative-based collaborative writing tool for constructing coherent technical documents*. PhD thesis, University of Southampton, 2007.

[3] Duvall, P., Matyas, S., and Glover, A. *Continuous Integration: Improving Software Quality and Reducing Risk (The Addison-Wesley Signature Series)*. Addison-Wesley Professional, June 2007.

[4] Ellis, C. A. and Gibbs, S. J. Concurrency Control in Groupware Systems. 18:399–407, May 1989.

[5] Henderson, P. and De-Silva, N. A narrative approach to collaborative writing: A business process model. In *8th International Conference on Enterprise Information Systems (ICEIS)*, Cyprus, 2006.

[6] Kermarrec, A.-M., Rowstron, A., Shapiro, M., and Druschel, P. The IceCube Approach to the Reconciliation of Divergent Replicas. In *Proceedings of the ACM Symposium on Principles of Distributed Computing - PODC 2001*, pages 210–218, Newport, Rhode Island, USA, August 2001. ACM Press.

[7] Lowry, P. B., Curtis, A., and Lowry, M. R. Building a Taxonomy and Nomenclature of Collaborative Writing to Improve Interdisciplinary Research and Pratice. *Journal of Business Communication*, 41(1):66–99, January 2004.

[8] Mann, W. C. and Thompson, S. A. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[9] Molli, P., Skaf-Molli, H., Oster, G., and Jourdain, S. SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments. In *Proceedings of the Conference on Computer-Supported Cooperative Work in Design - CSCWD 2002*, pages 80–85, Rio de Janeiro, Brazil, September 2002.

[10] Noël, S. and Robert, J.-M. Empirical study on collaborative writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work - JCSCW*, 13(1):63–89, 2004.

[11] Oster, G., Urso, P., Molli, P., and Imine, A. Tombstone transformation functions for ensuring consistency in collaborative editing systems. In *The Second International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*, Atlanta, Georgia, USA, November 2006. IEEE Press.

[12] Rahhal, C., Skaf-Molli, H., Molli, P., and Silva, N. D. Semcw: Semantic collaborative writing using rst. In *The 3rd International Conference on Collaborative Computing:Networking, Applications and Worksharing - CollaborateCom 2007*, New York, USA, nov 2007.

[13] Skaf-Molli, H., Ignat, C.-L., Rahhal, and Molli, P. New Work Modes for Collaborative Writing. In *International Conference on Enterprise Information Systems and Web Technologies- EISWT 2007*, Orlando, Florida, jul 2007.

[14] Skaf-Molli, H., Molli, P., and Oster, G. Semantic Consistency for Collaborative Systems. In *Proceedings of the International Workshop on Collaborative Editing Systems - CEW 2003*, Helsinki, Finlande, September 2003.

[15] Sun, C., Jia, X., Zhang, Y., Yang, Y., and Chen, D. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, March 1998.

[16] Tudor Groza, K. M., Siegfried Handschuh and Decker, S. SALT - Semantically Annotated LaTeX for scientific publications. In *4th European Semantic Web Conference (ESWC 2007).*, jul 2007.