SemCW: Semantic Collaborative Writing using RST

Charbel Rahhal, Hala Skaf-Molli, Pascal Molli, Nishadi Desilva

SemCW: Semantic Collaborative Writing using RST

Charbel Rahhal, Hala Skaf-Molli and Pascal Molli
LORIA-INRIA Lorraine
University Henri Poincaré Nancy, France
Email:rahal@loria.fr, skaf@loria.fr,molli@loria.fr

Nishadi De Silva
School of Electronics and Computer Science
University of Southampton, UK
Email: n.desilva@ecs.soton.ac.uk

***Abstract*—During collaborative writing each author works on a copy of the shared document. These copies are then merged to produce the final document. This asynchronous work is supported by several collaborative writing tools. While these tools are excellent at merging and detecting *syntactic* conflicts, they are not able to easily recognise *semantic* inconsistencies. This hinders the coherence of the document because while each individual copy might be well constructed, they may not be after the merge. To address this, we investigate the combination of the Rhetorical Structure Theory with Operational Transformation approach. In this paper, we define a data model, a set of operations to manipulate the RST structures and a set of transformation functions. A validity checker alerts the authors to areas in the text with possible semantic lapses in the merged documents.**

I. Introduction

Collaborative writing is the process by which several authors work on a document together. The major benefits of collaborative writing include reduced task completion time, reduced errors, and getting different viewpoints and skills [21], [11]. Various modes of working exist [21] depending on the proximity and synchronicity of collaborative work. Some groups all work in the same location and on the same time schedule. Other groups work on different schedules and the members may be geographically dispersed. This is common in technical writing scenarios such as the production of research papers by scientists in different countries or institutions.

The disadvantages of collaborative writing include difficult group coordination [11] and documents that are poorly structured[14]. The lack of structure usually arises from misaligned contributions by individual authors. While each section may be well constructed, they may not 'fit' logically when placed together. This is what is referred to as **semantic inconsistency** in this paper. Worse still, it is often not easy to detect such inconsistencies; thus making many collaboratively authored documents incoherent.

One way of assisting collaborative writing is by using software. There are various tools available today that enable teams of authors to create, update and merge documents [17]. We focus, in particular, on the optimistic replication model since it can support all collaborative interaction modes [17]. In this model, each author has his own copy of the shared data. This has many advantages such as achieving high responsiveness, preserving privacy [20] and enabling parallel working [10]. The optimistic replication algorithms deal with concurrency control problems and syntactic inconsistencies. The algorithms ensure that copies of the shared data converge towards a unique value. Therefore, all the authors will have the same value when the system is idle (i.e. no operations in the pipe). While this is important, it does not guarantee that the resulting text is coherent (or semantically sound) [2].

This is true for most of the work done so far in this field; they concentrate mainly on syntactic consistency. Only a few researchers have started handling semantic consistency problems [18], [6]. In their work, integrity constraints were used to ensure semantic coherence. However, capturing the semantics of a textual document through logical constraints has not been obvious. We realised that special relationships may be needed to define the coherence of a document.

Semantic consistency[1] is a subjective phenomenon. However, for the purposes of this research, we need to narrow down a definition. The ease with which a text can be read and understood can be influenced by several factors such as the grammar, the language and the previous knowledge of the reader. However, assuming all these criteria are fulfilled, it is still possible for a text to not make much sense. Several researchers [5], [8] have established that the coherence of a text is linked to its internal logical structure.

The mere sequence in which the sentences are positioned can influence how the paragraph is interpreted [7], [9]. For instance, take the two texts below. The first one is easy to understand. However, by just altering the order of the last two sentences, Text 2 has being made less comprehensible. This is because readers tend to assume logical connections between pieces of text in juxtaposition. It is necessary to make these logical connections as easy to determine as possible [23]. Otherwise there is an unnecessary burden placed on the reader.

$Text1$:

Semantic coherence is vital for an effective document. However, current tools do not provide support in this aspect of writing. Therefore, the combination of merging algorithms and RST is a significant step towards bridging this gap.

$Text2$:

[1] 'Semantic consistency' and 'coherence' are used interchangeably in this paper

Semantic coherence is vital for an effective document. Therefore, the combination of merging algorithms and RST is a significant step towards bridging this gap. However, current tools do not provide support in this aspect of writing.

Problems in a short text like the above are easy to identify and fix. In fact, we compose such texts daily in our conversations and e-mails without giving it much thought. However, the problem is much harder to solve in larger documents, particularly when multiple authors are involved.

Linguists who studied the structure of texts have attributed coherence to implicit logical relationships that exist between parts of a text. For instance, part A "provides background information" to part B, part D "provides evidence" to the claim made in part B and so on. So, it is important for authors to establish what these links are before they write and equally important they that convey them to the readers. Linguists also developed theories that enabled authors to study and analyse the logical structure of texts. We believe that using ideas from such a theory will greatly benefit the semantic aspects of collaborative writing. It is support for this aspect of writing that we find missing in collaborative writing tools and is the focus of our work.

In this paper, we provide a brief tutorial on Rhetorical Structure Theory (RST) which is our chosen discourse theory to address semantic consistency. In the section III, we define a data model and a set of operations to edit RST structures. In the section IV, we describe the fundamental principles of merging algorithms through two scenarios. In the section V, we outline Operational Transformation (OT) which is the merging technique we intend to use. In the section VI, we define transformation functions for RST operations. We finally present a possible visual representation of the detection of semantic inconsistency, followed by our conclusions.

II. Rhetorical Structure Theory (RST)

There are several discourse theories that have been developed to analyse the structure and coherence of text. We have chosen Rhetorical Structure Theory (RST) [8] for its simplicity and precise relationship definitions. This section gives a brief description on how to analyse a text using RST and discusses some aspects of the analysis relevant to this paper.

A. Analysing a text using RST

The first step in a RST analysis is to divide the text into non-overlapping segments. Each segment should have independent functional integrity and is often a clause [8]. As an example, Text 1 above has been divided into three segments as shown.

[**Text 1:**] [*Semantic coherence is vital for an effective document.*]^{1A} [*However, current tools do not provide support in this aspect of writing.*]^{1B} [*Therefore, the combination of merging algorithms and RST is a significant step towards bridging this gap.*]$1C$

The next step is to identify logical relationships that exist between pairs of segments. For instance, in the above example, we believe there is a Background relationship between segments A and B (i.e. segment A provides background information to understand segment B). Some relationships are often identified using cue words such as 'however', 'although' and so on [7], while others can be detected without any such phrases. However, in our application of RST the analyst is also often the author of the text. Therefore, having created the text with a certain understanding of it, we do not anticipate there to be major difficulties in recognising the RST relationships.

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. So, in our BACKGROUND (cf Figure 1) relationship, segment $1A$ is the satellite and segment $1B$ is the nucleus. Such relationships involving a nucleus and satellite are called **hypotactic** and are illustrated as below. Note that the arrowhead points towards the nucleus.

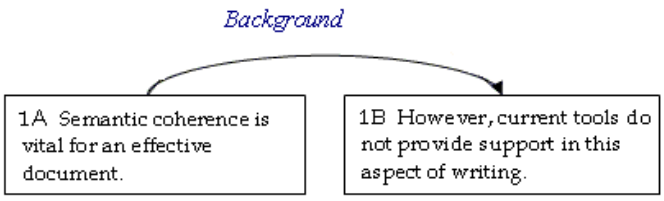


Fig. 1. Segment A and B have a Background relationship between them

A few relationships apply to segments of equal importance (e.g. Sequence, Contrast) and are called **paratactic**. In Mann and Thompson's paper (1988), there are 23 relationships defined. Each relationship has precise definitions for the nucleus, satellite and what their combined effect should be on the reader. Henderson and De Silva [4], however, considered 23 to be too many and began selecting a subset of relationships that were sufficient for analysing technical documents. In De Silva [1], a user study has shown that technical authors found a set of nine relationships adequate for their analyses.

In the analysis, segments involved in a relationship collectively form a **span**. A span can in turn become part of another relationship as shown below. The span of segments $1A$ and $1B$ is identified as being in a Motivation relationship with segment $1C$ in our example (i.e. the importance of semantic coherence and the lack of support for it has, together, has motivated us to combine existing ideas to solve the problem). Hence, the analysis is a recursive process and continues until all the segments are assembled into a tree of relationships as shown below (called a **RS-tree**).

Mann and Thompson (1988) conjecture that producing a well-formed RS-tree for a text indicates that a text is coherent. They define four properties that determine if a RS-tree is well formed. They are:

- ***Completedness*** One schema application (the root) should

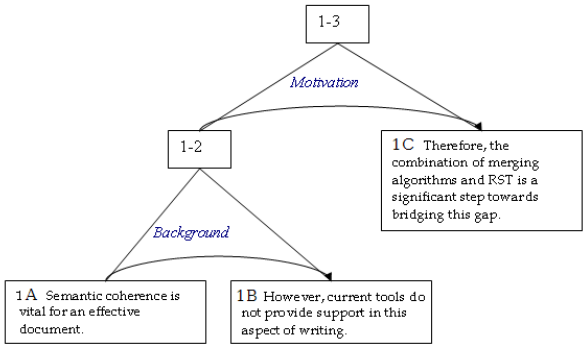


Fig. 2. A possible RS-tree for Text 1

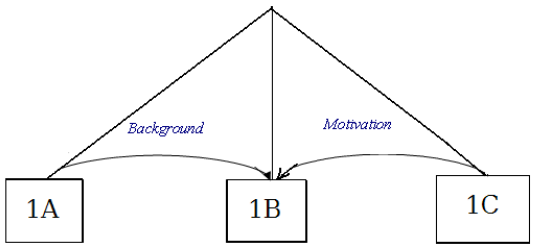


Fig. 3. An n-ary RS-tree

cover the entire text.

- ***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.
- ***Uniqueness*** Each text span/segment should have only one parent (i.e. each schema application consists of a different set of text spans/segments).
- ***Adjacency*** Only adjacent text spans/segments can be grouped together to form larger spans.

We make use of these properties to test for consistency after changes to a document are merged.

B. The role of RST in collaborative writing

We propose that the co-authors agree on the RST relations between the various sections of the document at the start. This in itself is a useful exercise to iron out differences in the understanding of the 'story' that their document ought to convey. For instance, authors could have differing opinions about what roles the sections should play. Having a well-formed RS-tree for the document also gives the authors some confidence that their document is coherent. Each author then starts work on his assigned section with an *understanding* of how that section fits in with the rest of the document. This is perhaps the greatest benefit of using RST in this context.

There is a significant difference between this method and traditional applications of RST. Usually, RST is applied to a 'static' text. However, in collaborative writing, the text continually changes. As the text changes, existing RST relationships may no longer apply. Authors too can decide to change the analysis to something they believe is better. In order to accommodate this behavior, we define a validity checker as described in section VI-C.

C. Converting n-ary RS-trees to binary trees

In the analysis above, the RS-tree produced was binary. However, it is common to have n-ary RS-trees (see figure 3).

For the purposes of our research, we restrict RS-trees to be binary. This has been done by other researchers too [9] and makes computations and software implementations easier.

The n-ary tree above can easily be converted into a binary tree (cf figure 4).

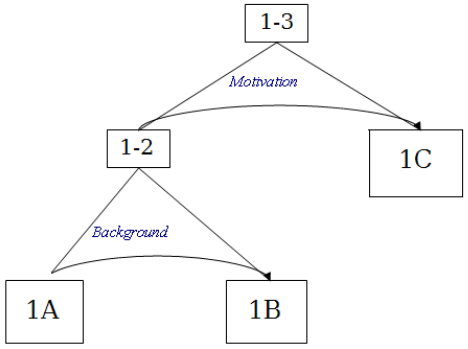


Fig. 4. A binary RS-tree

There are some decisions that need to be made as to which of the two relationships will appear on top. In this case, we have chosen to have the Motivation on top and the Background at the bottom. It could have been the other way round too. However, in both cases, the sequence of nodes remain the same (i.e. doing a pre-order traversal) and the broad logical structures in both are comparable. In fact, having to make such decisions can be useful too. In our example: Is section C going to provide motivational information for section B only or for both sections A and B, together? So, we do not envisage this to be a problem in our application and continue to use binary trees.

D. Representing RS-trees using URML

To make RST-annotated corpus [2] data readable by both humans and computers, an XML format called URML (Under-specified Rhetorical Markup Language) [15] was introduced. The benefit of URML is that the entire RST analysis does not have to be known at the time the URML file is created

[2]Corpus: A collection of writings or recorded remarks used for linguist analysis

(hence, underspecified). URML allows the RST analysis to be developed incrementally.

As an example, figure 5 shows the URML representation for the RS-tree of figure 2.

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE urml SYSTEM "urml.dtd">
<urml>
 <header>
  <reltypes>
    <rel name="Background" type="hyp" />
    <rel name="Motivation" type="hyp" />
  </reltypes>
 </header>
 <document id="sample001">
  <text>
   <segment id="1A"> Semantic coherence is vital
   for an effective document.
   </segment>
   <segment id="1B"> However, current tools do
   not provide support in this aspect of writing.
   </segment>
   <segment id="1C"> Therefore,
   the combination of merging algorithms and
   RST is a significant step towards bridging
   this gap.</segment>
  </text>
  <analysis status="forest-complete">
   <hypRelation id="1-2" type="Background">
     <satellite id = "1A" />
     <nucleus id ="1B"/>
   </hypRelation>
   <hypRelation id="1-3" type="Motivation">
     <satellite id= "1-2" />
     <nucleus id ="1C"/>
   </hypRelation>
  </analysis>
 </document>
</urml>
```

Fig. 5. URML representation for the RST analysis of Text 1

III. Storing and editing RS-trees

This section shows how RS-trees can be represented in a data model and also introduces the set of operations that allows authors to edit them.

A. A data model

Looking at the URML above, it becomes clear that a RS-tree is an ordered sequence of text segments and a set of RST relationships. In order to model RS-trees, we create a data model to store these elements: Segment and Relation.

A *segment* has the following attributes:

- *Position* is the position of this segment in the document. In order to maintain coherence, it is important that the segments are in the correct order.
- *ID* is a unique identifier for each segment given by the system.
- *Content* is the textual content of each segment.
- *Visible* is a boolean attribute which is true by default and turns to false when the segment is deleted.

A segment can also be used to represent a span (an ordered sequence of adjacent segments).

Similarly, a *relation* has the following attributes:

- *ID* is a unique identifier for the relation given by the system.
- *Nucleus* is the identifier of the segment that is the nucleus in this relationship.
- *Satellite* is the identifier of the segment that is the satellite (or second nucleus) in this relationship.
- *Name* is the name of the relation such as Motivation or Background.
- *coveredSegments* is a set that contains the positions of the satellite and the nucleus of the relation. It is used in the detection of the violation of the rhetorical properties.

Using these two data elements, we proceed now to develop four operations that can be performed on RS-trees.

B. Operations on RS-trees

We introduce the four operations below that can be used to manipulate the RS-trees.

- $addSegment(position, id, content, sid)$ adds a segment in the specified position in the document. sid is the identifier of the site that generates this operation.
- $delSegment(position)$ this operation deletes logically a segment at the given position. This means that the segment is marked as invisible. There is no physical deletion for segments to ensure convergence during data reconciliation as we shall see later.
- $addRelation(Rid,\ from{<}type,\ id'{>},\ to{<}type,\ id"{>},\ Rname)$ operation adds a relationship across the specified segments and groups them into a span. The parameter *type* is used to indicate if the segment is a nucleus or a satellite.
- $delRelation(id)$ this operation deletes the specified relation from the relations set.

IV. Merging changes to RS-tree

In this section, we describe the fundamental principles of merging algorithms. These algorithms are based on optimistic replication approach [16]. This approach considers n sites e.g. a user in our context. There is a total order on the identifier of these sites. Each site has a copy of shared data e.g. the URML file in our context. A site modifies its local copy by producing an operation e.g. "add a segment at a given position" in our context. This operation is:

1) executed locally,
2) broadcasted to other sites,
3) received by others sites,
4) integrated locally.

The system is correct if all copies are identical when the system is idle.

In the following sections, we present two scenarios where two authors are modifying in parallel a shared URML document. The URML structure looks like a tree, but in fact we

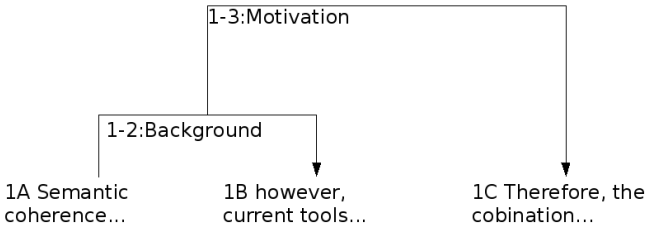


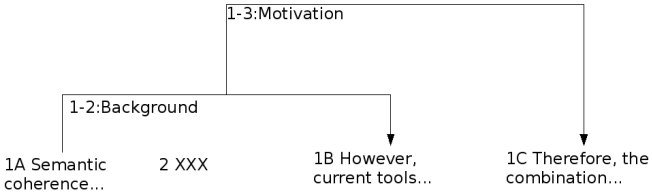
Fig. 6. URML data revisited : Initial state

have basically a sequence of segments that represents the text, and a set of relations. We can illustrate the URML data as in figure 6.

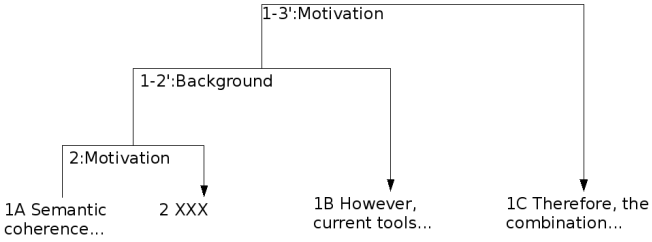
A. Concurrent adding scenario

In this scenario, we assume that u_1 and u_2 are editing the $Text1$ given in the section I. Each author has his own copy of the $Text1$. They both agree on the RST analysis given in figure 6.

Suppose now that u_1 adds a new segment identified by '2' containing the text "XXX" in position 2, between segment '1A' and segment '1B'. The result on the copy of u_1 is as below:



This RS-tree is inconsistent, it violates the connectedness property. The author u_1 has to change his RS-tree as below:

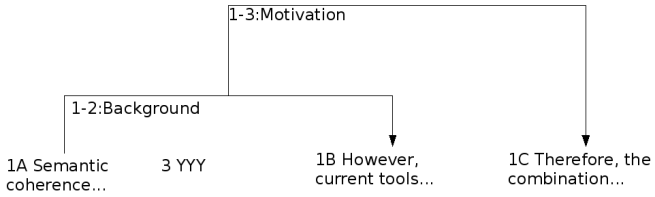


A new relation Motivation has been added by u_1 between '1A' and new segment 2. Old relation "1-2" has been deleted by u_1 and replaced by a relation "1-2'" linking relation 2 and segment '1B'. As old relation "1-2" has been deleted by u_1, relation "1-3" has been deleted by u_1 and replaced by a relation "1-3'". This way of updating relations is very important for the merging process. This means that the changes performed by u_1 have been detected by the system as the following sequence of operations P_1 :

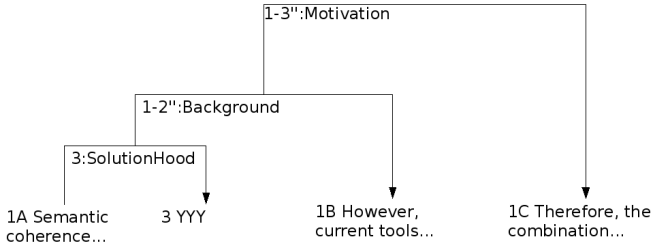
Of course, we can imagine another scenario where relations are not deleted but just updated. Unfortunately, this scenario is difficult to achieve because operations are traditionally detected using diff algorithms and such algorithms often detect updates as delete operation followed by insert operations.

```
P1={
addSegment(position=2,id=2,content="XXX");
delRelation(1-3);
delRelation(1-2);
addRelation(id=2,<S,1A>,<N,2>,"Motivation");
addRelation(id=1-2',<S,2>,<N,1B>,"Background");
addRelation(id=1-3',<S,1-2'>,<N,1C>,"Motivation");
}
```

At the same time, the author u_2 on the $site2$ performs concurrent operations. He adds a new segment identified by '3' containing the text "YYY" in position 2 e.g. between segment '1A' and segment '1B'. The result on the copy of u_2 is:



As this state is inconsistent, u_2 adapts the RS-tree as below:



The author u_2 produces the following set of operations:

```
P2={
addSegment(position=2,id=3,content="YYY");
delRelation(1-3);
delRelation(1-2);
addRelation(id=3,<S,1A>,<N,3>,"SolutionHood");
addRelation(id=1-2'',<S,3>,<N,1B>,"Background");
addRelation(id=1-3',<S,1-2'>,<N,1C>,"Motivation");
}
```

The execution of the first step of the scenario is illustrated in figure 7. u_1 has generated P_1 and u_2 has generated P_2, so the copies hosted by u_1 and u_2 are now diverging. Both sites exchange their operations and run the integration process.

In order to converge, the system has to ensure that $Merge(P1, P2) = Merge(P2, P1)$. Unfortunately, this property is not ensured by traditional merge algorithms. Divergence will occur on segments as depicted in figure below:

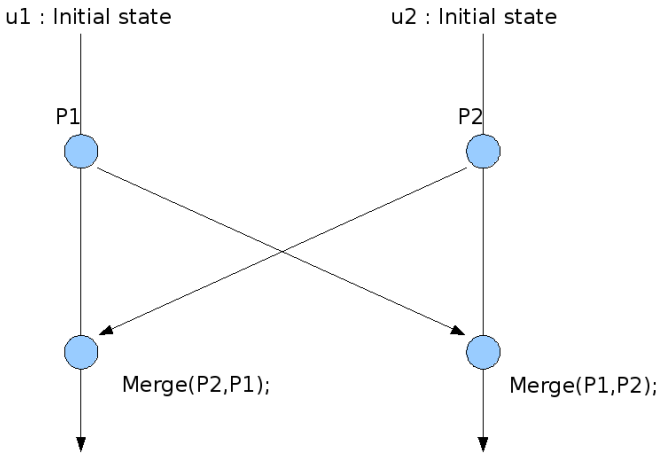
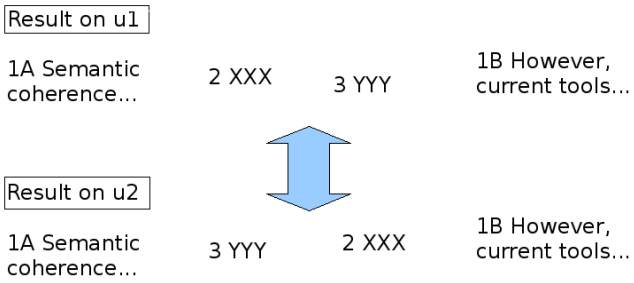


Fig. 7. Global merging scenario

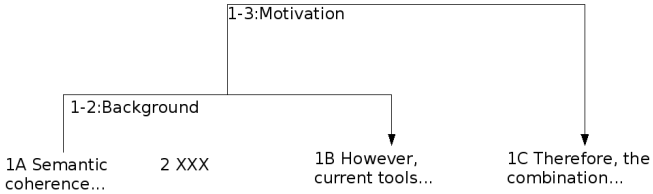


This problem is well-know in CSCW community. The Operation Transformation (OT) framework [3] has been developed to ensure convergence in these conditions.

B. Concurrent add-delete scenario

In this scenario, we assume two authors u_1 and u_2 are editing the initial $Text1$. User u_1 adds concurrently a new segment and changes the relations, while user u_2 deletes a segment.

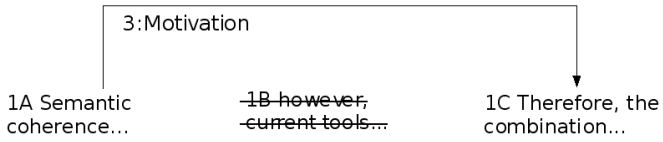
Suppose that u_1 adds a new segment identified by '2' containing the text "XXX" in position 2, between segment '1A' and segment '1B', as in the first scenario. Therefore, he produces the same sequence of operations $P1$. The result on the copy of u_1 is as below:



At the same time, the author u_2 concurrently deletes the second segment identified by '1B' and adds a new relation Motivation between 1A and 1C. He produces the following set of operations:

```
P2={
delSegment(1A);
delRelation(1-3);
delRelation(1-2);
addRelation(id=3,<S,1A>,<N,1C>,"Motivation");}
```

The result on the copy of u_2 is as below:



V. Operational Transformation approach (OT)

The Operational Transformation (OT) approach [3] is an optimistic replication model used in the real-time group editors domain. It is a *theoretical framework* that allows to build a generic and safe synchronizer [10]. OT considers n sites where each site has a unique identifier and owns a copy of shared data. There is a total order on the sites. When a site performs an update, it generates a corresponding operation, which is first executed locally and then broadcasted to the other sites. Every operation is processed in four steps: (a) generated on one site, (b) broadcasted to the other sites, (c) received by the other sites, (d) executed on the other sites.

The execution context of a received operation op_i (step c) may be different from its generation context (step a). In this case, the integration of op_i by other sites may lead to inconsistencies between the replicas of data. As an example, we consider two sites - $site_1$ and $site_2$ - working on a shared data of type *string of characters* initially equal to the string "efect". A string of characters can be modified with the operation *ins(p,c)* for inserting a character c at position p in the string. We assume the position of the first character in a string is 0. $User_1$ and $user_2$ generate and execute two concurrent operations op_1=*ins(2,f)* and op_2=*ins(5,s)*, respectively. When op_1 is received and executed on $site_2$, it produces the expected string "effects". However, when op_2 is received on $site_1$, its execution leads to the state "effecst" since it does not take into account that op_1 has been executed before it. In the end, the copies of $site_1$ and $site_2$ do not converge.

In the operational transformation (OT) approach, before being executed, received operations are transformed according to concurrent operations that have already been executed on the local copy. This transformation is performed by calling the appropriate **transformation functions**.

Definition A transformation function T takes two concurrent operations, op_1 and op_2, that must be defined on the same state S. The function computes a new operation op'_1 equivalent to op_1 (e.g. has the same effects) but defined on the state $S' = S \odot op_2$. S' is the state resulting from the execution of op_2 on state S.

Using OT approach, our previous example is now executed as follows. When op_2 is received on $site_1$, op_2 needs to be transformed according to the previously executed operation, op_1. The integration algorithm calls the transformation function $T(op_2$=*ins(5,s)*,op_1=*ins(2,f)*$)$ = *ins(6,s)* = op'_2.

The insertion position of op_2 is incremented since op_1 has inserted an f before s in state "efect". After the execution

of op'_2, the state of $site_1$ becomes "effects". On the contrary, when op_1 is received on $site_2$, the transformation does not modify op_1's parameters since f is inserted before s. Thus, op_1 is executed as-is and the state of $site_2$ is "effects". In this scenario, OT approach has ensured that both copies converge to the same value.

The OT approach distinguishes two main components: an *integration algorithm* and a set of *transformation functions*. The integration algorithm is in charge of reception, diffusion and execution of operations. When necessary, it calls transformation functions. This algorithm does not depend on the type of replicated data. The transformation functions merge concurrent modifications by serializing two concurrent operations. These functions are specific to a particular type of replicated data such as a string of characters [10], XML documents[12], calendars or file systems.

OT approach aims to achieve *convergence* of copies.

a) Convergence: Like every optimistic replication algorithm, the OT approach aims to ensure eventual consistency. This means that if no updates are performed for a long period of time, all updates will eventually propagate through the system and all the copies will converge towards the same value. In other words, when the system is idle (no operation in pipes), all the copies become identical.

To ensure convergence, it has been proved [19] that the underlying transformation functions must satisfy two properties:

Definition The TP_1 property defines a *state equivalence*. The state generated by the execution of op_1 followed by $T(op_2, op_1)$ must be the same as the state generated by the execution of op_2 followed by $T(op_1, op_2)$: $op_1 \circ T(op_2, op_1) \equiv op_2 \circ T(op_1, op_2)$

Definition The TP_2 property ensures that the transformation of an operation regarding a sequence of concurrent operations does not depend on the order in which operations of this sequence were transformed: $T(op_3, op_1 \circ T(op_2, op_1)) = T(op_3, op_2 \circ T(op_1, op_2))$

The OT approach could be used to design a reconciliation framework able to reconcile divergent copies of any type of data. In order to build such a framework, the following tasks have to be completed. First, an integration algorithm must be chosen; regarding this algorithm, TP_2 property may be required on underlying transformation functions. Secondly, operations which could be performed on shared data types must be defined. Finally, the required transformation functions for all combinations of operations have to be provided.

We have already used OT to synchronize text [10] and XML documents [12]. In this paper, we will use OT to synchronize RS-trees.

VI. SemCW: Merging RST data with OT

OT can be used to manage efficiently the merging of RS-trees. Segments are ordered as a sequence.

A. RST Transformation functions

The following transformation functions deal with concurrent segments operations and ensure convergence.

$T(addSegment(n_1, id_1, v_1, sid_1), addSegment(n_2, id_2\ v_2, sid_2))$ =
  if $(n_1 < n_2)$ **or** $(n_1 = n_2$ **and** $sid_1 < sid_2)$
    return $addSegment(n_1, id_1, v_1, sid_1)$
  else return $addSegment(n_1 + 1, id_1, v_1, sid_1)$
  endif

The above function transforms $op_1 = addSegment(n_1, id_1, v_1, sid_1)$ regarding $op_2 = addSegment(n_2, id_2, v_2, sid_2)$. The main idea is to compare the insertion position of two concurrent addition of segments in the sequence of segments.

- If op_1 inserts a segment at a position after the insertion position of op_2 then the insertion position of op_1 has to be shifted one position to the right. Therefore, its insertion position is incremented.
- If op_1 inserts a segment before the insertion position of op_2 then the insertion position of op_1 remains the same.
- If op_1 and op_2 insert a segment at the same position, T must decide the serialization order. In the above definition, the decision of T is based on the site identifier. If the site identifier of op_1 is greater than the site identifier of op_2 then the insertion position of op_1 is shifted one position to the right, else it remains the same. Of course, this an arbitrary choice.

$T(delSegment(n_1), delSegment(n_2))$ =
  if $(n_1 = n_2)$
    return id()
  else return $delSegment(n_1)$
  endif

This function transforms $op_1 = delSegment(n_1)$ regarding $op_2 = delSegment(n_2)$. If op_1 and op_2 delete the same segment, then the function T disables effect of op_1 by transforming it into an identity operation. Else T returns op_1. In order to ensure the correctness of our transformation functions (that they satisfy *TP1* and *TP2* properties), we use the *TTF* approach [13]. We keep the deleted segment as a tombstone, this means that we keep the segment in its position and mark it as invisible. Consequently, deleted segments must remain present in the model, but are hidden from the user.

$T(delSegment(n_1), addSegment(n_2, id_2, v_2,$
$sid_2))$ =
  if $(n_1 < n_2)$
    return $delSegment(n_1)$
  else return $delSegment(n_1 + 1)$
  endif

$T(addSegment(n_1, id_1, v_1, sid_1), delSegment(n_2))$
=
return $addSegment(n_1, v_1, sid_1)$

The above transformation functions $T(delSegment, addSegment)$ and $T(addSegment, delSegment)$ are easy to understand.

In fact, the transformation functions for all the remaining couples of operations (e.g. $T(delSegment, addRelation)$,

$T(addRelation, delSegment), \ldots)$ return the operation itself. Because according to our data model, the relation part of RS-tree is just a set of relations. Therefore, it is impossible to generate *syntactic* conflicts by adding or removing relations. Conflicts will occur only at the semantic level.

We need only to define the following transformation function because it is impossible to delete the same relation twice.

```
T(delRelation(id1), delRelation(id2)) =
  if (id1 = id2)
      return id()
  else return delRelation(id1)
  endif
```

B. Scenarios with OT

Now, if we apply OT approach with our transformation functions to the first scenario in the section IV, both sites will converge to the value represented in the figure 8.

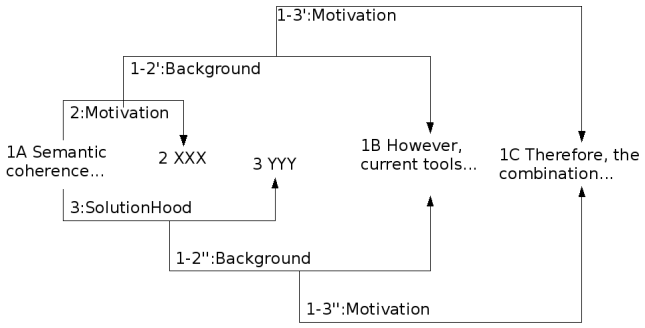


Fig. 8. URML convergent final state in scenario1

In the same way, both sites in the second scenario in the section IV will converge to the value represented in the figure 9

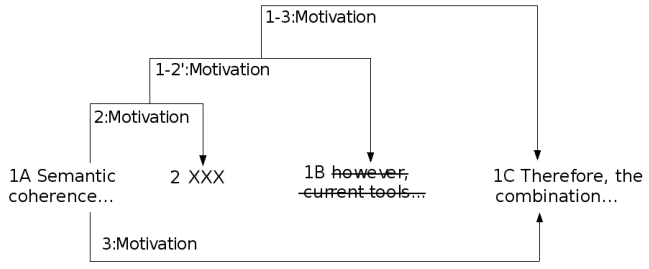


Fig. 9. URML convergent final state in scenario2

As we can see, the final states has both RS-trees, this will help the authors to better understand the reasons of the conflicts. The syntactic convergence is ensured, however, semantic consistency is not ensured. The result does not respect the rhetorical properties. To solve this problem, we developed a validity checker as described in the next section.

C. Validity checker

The merge of two *well formed* RS-trees is not necessarily a *well formed* RS-tree.

The main interest of this validity checker is to detect the semantic inconsistency of the document. In other words, the validity checker will detect the violation of the rhetorical properties and informs the user about them. We start by presenting the definition of some elements needed by the validity checker:

- *Segments* is the ordered sequence of the text segments.
- *Relations* is the set of all the relations in the RS-trees.
- *ConnectViol* is the set of the segments violating the connectedness property.
- *UniqViol* is the set of the segments that violate the uniqueness property.
- *AdjViol* is the set of the relations that violate the adjacency property.
- *AllTxtSegments* is the set containing all the segments' positions of the document.
- *coveredSegments* is a set of segments' positions covered by a relation. For example, if a relation R has a satellite at a position 1 and a nucleus at a position 2, then R.coveredSegments =$\{1,2\}$. If a relation R has a relation R' as satellite and a segment at position 6, then R.coveredSegments = R'.coveredSegment $\bigcup \{6\}$.

The validity checker detects the violation of each property in a simple formal way:

1) *Connectedness violation*
   $(\forall$ S $\in$ Segments) $[\nexists$ R $\in$ Relations : R.Nucleus = S.ID $\bigvee$ R.Satellite = S.ID$] \Rightarrow$ S $\in$ ConnectViol.
2) *Uniqueness violation*
   $(\forall$ S $\in$ Segments) $[\exists$ R, R'$\in$ Relations: R $\neq$ R' $\bigwedge$ (R.Satellite = R'.Satellite = S.ID $\bigvee$ R.Satellite = R'.Nucleus = S.ID $\bigvee$ R.Nucleus = R'.Nucleus = S.ID $\bigvee$ R.Nucleus = R'.Satellite = S.ID)$] \Rightarrow$ S $\in$ UniqViol.
3) *Adjacency violation*
   $(\forall$ R $\in$ Relations) :(R.Satellite.position>R.Nucleus.position+1) $\bigvee$ (R.Nucleus>R.satellite+1) $\Rightarrow$ R $\in$ AdjViol.
4) *Completedness violation*
   $(\exists$ S $\in$ Segments)$\{\exists$ R $\in$ Relations : S.position $\notin$ R.coveredSegments $\} \bigvee \{\exists$ R, R' $\in$ relations : R $\neq$ R' $\bigwedge$ R.coveredSegments = R.coveredSegments $\bigwedge$ coveredSegments = AllTxtSegments)$\}$

The checker validity indicates the incoherence parts of the document. It provides the users an awareness about what happened. Based on this knowledge, users are able now to solve the semantic conflicts in a better way.

For example, if we apply the validity checker to the figure 8, it will traverse the ordered sequence of segments and the set of the relations and checks the violation of each rhetorical property. It will present the following inconsistency:

- Two relations starting from the first segment which violates the uniqueness property.
- The second segment is connected in one tree and disconnected in an another tree.

- The third segment is connected in one tree and disconnected in an another tree.
- The solutionHood relation violates the adjacency property.
- The 1-2': Background relation violates the adjacency property.
- Two relations sibling the fourth segment which violates the uniqueness property.
- Two relations sibling the last segment which violates the uniqueness property.
- Neither the first tree, nor the second tree respects the completeness property. There exists one segment in each tree not covered by its root.

A possible visualization of the validity checker for the example in figure 8 is shown below:

The user has to resolve the conflicts manually. Either by deleting XXX (with the relations) or by deleting YYY (with the relations) or by leaving both texts and replacing the majority of the relations like he is doing a new analysis of the merged text.

VII. Related Work

In the collaborative writing domain, most work on semantic consistency are based on constraints. Skaf-Molli et al. [18] propose to integrate these constraints with the OT approach in order to ensure semantic consistency in merged XML documents. If the constraints are violated after merging, the problem is fixed by adding or deleting some actions. In [6], semantic consistency is handled as a constraints optimisation problem. If an operation violates the constraints, the operation is canceled. Both the approached in [18] and [6] bring about lost updates which is not ideal.

Moreover, we are convinced that sometimes it is not possible to capture semantic knowledge by means of constraints. Constraints can define structural elements such as a document having only one title. However, they cannot capture the co-author's understanding and logical reasoning about the text. Finally, with the constraints approach, the constraints are *outside* the document. However, with the RST approach, the semantic annotations are part of the document. Therefore, when authors exchange documents, they also pass on their understanding of it via the attached RST relationships.

In previous work regarding coherence in collaborative writing [2], RST was used to analyze an executive summary like outline of the document (called a *document narrative*). This technique is called narrative-based writing [1] and enhances the implicit story conveyed by a document to the readers; thereby improving coherence. In this paper, we explore the idea of applying RST directly to the document. The RST relationships help the authors to see the ways in which the sections depend on each other.

The project SALT (Semantically Annotated Latex) has some common features with our work. In [22], 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.

In our work, RST is not just used to add semantic annotations within the document but also as a tool to evaluate the document's level of coherence. By constantly maintaining g and checking the four properties, we are able to detect inconsistencies and alert the authors to such areas. We anticipate that we can integrate our work easily into the SALT framework such that our RST capability can be extended into LaTex documents too.

VIII. Conclusion and Future Works

This paper described SemCW approach which is the combination of ideas from RST and the OT approach. SemCW allows to detect semantic inconsistency in merged texts. We were motivated to carry out this research after observing a lack of semantic support in collaborative writing tools of today. We envisage that collaboratively produced documents will be annotated with RST relations. As changes are made to the text in the document, these relations need to be changed to communicate the shifts in the narrative goals of the document. Having this understanding of how the text 'works', in our opinion, is the greatest benefit of analyzing the document using RST. The co-authors can then write their individual sections which should, in theory, better fit together.

This paper presented a data model and a set of operations that are necessary for authors to manipulate RS-trees. We also defined transformation functions for each combination of operations so that concurrent operations can be integrated correctly. The validity checker allows to detect semantic inconsistency in the merged document. The visualization at the end showed a possible user interface for this approach.

We believe this work bridges a gap in existing collaborative writing tools. At the moment, these tools handle syntactic conflicts excellently but do not address semantic problems that arise as a result of misaligned contributions by the various authors. RST plays a pivotal role in encouraging authors to think about the underlying logical relationships in their document and the consequences of updates. We realize that appending RST information is not easy and can be seen as cumbersome. However, with more research and experimental evaluation, the process can be made simpler and seamlessly integrated into current tools. We are working now on the definition of an ontology based on the RST and we will use Web Ontology Language (OWL) to detect the violation of RST properties.

References

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

[2] N. De-Silva and H. Skaf-Molli. Narratives to preserve coherence in collaborative writing. In *The Eight International Workshop on Collaborative Editing, With CSCW 2006*, Banff, Canada, November 2006.

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

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

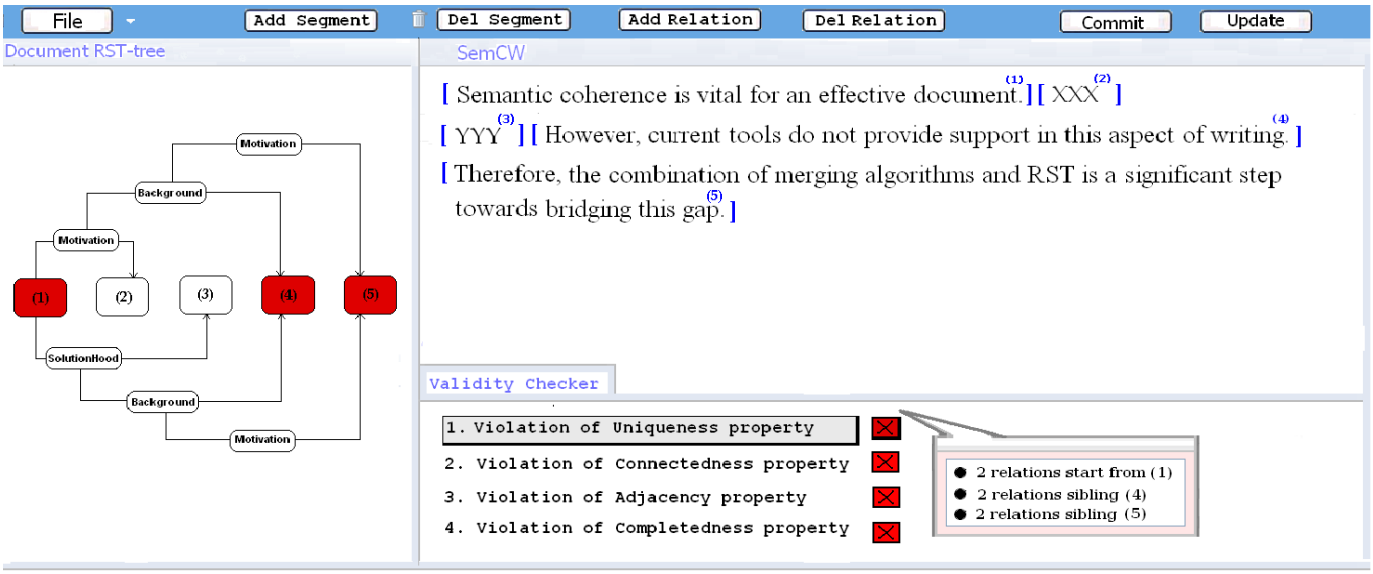


Fig. 10. The validity checker

[5] J. R. Hobbs. Towards an understanding of coherence in discourse. In W. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing*, pages 223–244. Lawrence Erlbaum Associates, Inc., New Jersey, 1982.

[6] A.-M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. 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] A. Knott. *A Data-Driven Methodology for Motivating a Set of Coherence Relations*. PhD thesis, University of Edinburgh, 1996.

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

[9] D. Marcu. *The Theory and Practice of Discourse Parsing and Summarization*. The MIT Press, 2000.

[10] P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003*, pages 212–220, Sanibel Island, Florida, USA, November 2003. ACM Press.

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

[12] G. Oster, H. Skaf-Molli, P. Molli, and H. Naja-Jazzar. Supporting Collaborative Writing of XML Documents. In *Proceedings of the International Conference on Enterprise Information Systems - ICEIS 2007*, Funchal, Madeira, Portugal, jun 2007.

[13] G. Oster, P. Urso, P. Molli, and A. Imine. 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.

[14] A. C. Paul Benjamin Lowry and M. R. Lowry. Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice. *Journal of Business Communication*, 41(1):66–99, 2004.

[15] D. Reitter and M. Stede. Step by step: Underspecified markup in incremental rhetorical analysis. In *Proceedings, 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, Budapest, 2003.

[16] Y. Saito and M. Shapiro. Optimistic Replication. *ACM Computing Surveys*, 37(1):42–81, 2005.

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

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

[19] M. Suleiman, M. Cart, and J. Ferrié. Concurrent Operations in a Distributed and Mobile Collaborative Environment. In *Proceedings of the International Conference on Data Engineering - ICDE'98*, pages 36–45, Orlando, Florida, USA, February 1998. IEEE Computer Society.

[20] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. 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.

[21] S. G. Tammaro, J. N. Mosier, N. C. Goodwin, and G. Spitz. Collaborative Writing Is Hard to Support: A Field Study of Collaborative Writing. *Computer-Supported Cooperative Work - JCSCW*, 6(1):19–51, 1997.

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

[23] J. Zobel. *Writing for computer science*. Springer, USA, 2 edition, 2004.