Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

A narrative-based collaborative writing tool for constructing coherent technical documents

by

Nishadi H. De Silva

A doctoral thesis submitted in partial fulfilment
of the requirements for the award of
Doctor of Philosophy

April 2007

To my parents

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

A narrative-based collaborative writing tool for constructing coherent technical documents

by Nishadi H. De Silva

One important feature of an effective document that makes it easy to read and understand is known as **coherence**. Technical documents produced collaboratively are often incoherent due to a lack of group consensus and misaligned contributions by the individual authors. However, current document planning techniques and writing tools do not provide explicit support for improving coherence. The goal of this research, therefore, is to develop and evaluate a new technique and tool that helps teams of authors to structure coherent technical documents.

The coherence of a document can be attributed to the story (or narrative) it conveys to the reader. If this story is consistent and coherent, the same can be said about the document. A discourse theory such as Rhetorical Structure Theory (RST) that has been developed by linguists helps further to analyse and improve a narrative. RST explains the coherence of a text by virtue of relationships (such as "paragraph A *justifies* paragraph B") between parts of the text. This research has combined the ideas from these parallel strands of research to develop a new document planning technique called **narrative-based writing**. The method involves writing down an explicit précis of the story (called a **document narrative or DN**) and then analysing it using RST. The DN and RST analysis are then used to structure the eventual document.

To extend the usability of narrative-based writing to geographically-dispersed authors, I have designed and implemented a collaborative tool that allows co-authors to edit, analyse and review DNs. The thorough design for the tool uses a combination of three models (conceptual, business process and functional) culminating in a set of functions that enable collaborative narrative-based writing. This dissertation discusses how, in the future, these functions could be incorporated in existing collaborative writing tools. Implementing this tool, albeit in its current prototypic state, has been invaluable in understanding the complexities of modelling and manipulating DNs and RST structures. Initial investigations using the new technique and tool have been positive, encouraging me to continue the research and evaluation in this field.

Acknowledgements

This research has been funded completely by a scholarship from the School of Electronics and Computer Science, University of Southampton, UK and by an Overseas Research Students (ORS) award (Universities UK).

Firstly, I am indebted to my PhD supervisor, Professor Peter Henderson, for the technical guidance and motivation to reach new heights. Thank you very much.

I am thankful to all my friends and colleagues who supported me throughout my PhD work.

I would also like to pay tribute to all my past school teachers and principals who went above and beyond their call of duty for their pupils.

Last but by no means least, I am eternally grateful to my family: to my parents for everything they have done for us; and to Prasadi for being a loving sister and loyal confidante.

List of publications

1. De Silva, N. and Skaf-Molli, H. (2006) Narratives to preserve coherence in collaborative writing. Proceedings of the 8th International Workshop on Collaborative Editing Systems, Banff, Canada.

2. Henderson, P. and De Silva, N. (2006) A narrative approach to collaborative authoring: A business process model. Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS), Cyprus.

3. De Silva, N. and Henderson, P. (2005) Narrative support for technical documents: Formalising Rhetorical Structure Theory. Proceedings of the 7th International Conference on Enterprise Information Systems (ICEIS), USA.

4. De Silva, N. (2005) A narrative approach to technical document construction. Proceedings of the PREP 2005 conference, Lancaster University, UK. (Awarded the EPSRC prize for best oral presentation in Computer Science)

5. De Silva, N. and Henderson, P. (2005) Computer Support for Narrative Structures. Proceedings of Computers and Writing 2005, Stanford University, USA.

Table of contents

Chapter 1

Introduction

> *Certain kinds of presentations, texts, have a kind of wholeness or integrity that others lack. We recognise that they "hang together" and are understandable as whole objects. They are coherent.*
>
> *(Mann et al., 1992)*

1.1 Background to the problem

The need to put things in writing could be, as Barbara Minto (2002) states in her book, "one of the least pleasant aspects of a professional person's job". It is not just the need to produce documents that is challenging but the requirement to produce *good* documents.

What is a good document? A good document should contain useful information for the reader. Furthermore, the style of the sentences, grammar, language and punctuation should be appropriate. However, a document in which these attributes are all flawless can still fail to make sense. The way in which the sentences are placed together can dramatically influence the reader's understanding of the text. This, often undervalued, characteristic of a document is called **coherence**. For a document to be good and effective, it *must* be coherent.

Achieving document coherence is not always straightforward. This is particularly true for **technical documents.** By technical documents we refer to a variety of forms of communication in a scientific context. Some examples include research papers, conference presentations, theses and websites. Technical documents are often written by authors who do not come from a linguistic background or have no formal training in writing (Kieras, 1989). They are also commonly produced by multiple authors working together. In this case, the possible lack of a group consensus and misaligned contributions by different authors can affect document coherence even further.

We found that document coherence can be attributed to the **story** conveyed to the reader in a document. Other researchers have made similar observations (Evans and Gruba, 2004, Zobel, 2004). Readers automatically search for relationships between the ideas that appear in

sequence (Minto, 2002) and deduce, perhaps even anticipate, the story. It is important, therefore, to plan this story prior to constructing the document. However, current techniques available for technical authors to organize the material in a document and collaborative writing tools do not provide sufficient support to formulate this story.

1.2 Outline of our solution

Our solution to this problem is as follows: We introduce a new technique called **narrative-based writing** and, develop and evaluate a tool that enables a team of writers engage in this technique.

Narrative-based writing draws together ideas from two parallel strands of research: narratives and technical writing. In particular, we have made use of a discourse theory developed by linguists called Rhetorical Structure Theory (RST). In narrative-based writing, authors are encouraged to write down a précis of the story their document will convey to the readers. This story is called the **document narrative (DN).** As an example, see the DN for this thesis in Figure 1-2.

Once the DN has been created, it can be analysed using RST to enhance and evaluate its coherence. RST explains coherence by asserting relationships between parts of the text (e.g. A *motivates* B). These relationships are illustrated as shown below. By identifying these relationships, authors are able to understand and improve the DN. Moreover, it is conjectured by the authors of RST that the text is coherent if these relationships can be assembled into a tree structure. This attribute of RST gives authors a mechanism to judge the quality of their DN before beginning to write.

The final part of the technique is producing the document. The sequence of parts in the DN dictate the sequence of sections in the document and the RST relationships give some guidance about the content of each of the sections.

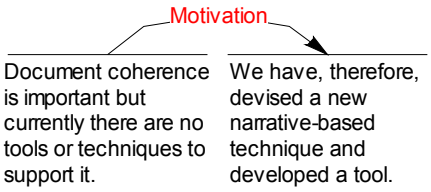


Figure 1-1: An example of a MOTIVATION relationship. The lack of support in existing tools and techniques motivated us to develop narrative-based writing and the tool.

In order to extend the use of narrative-based writing to teams of geographically-dispersed authors, we designed and built a Web-based tool. The most important aspect of implementing this tool was studying, in detail, the functions that were needed to collaboratively manipulate RST tree structures. These functions have been implemented in Java and have helped us understand the complex computational aspects of manipulating RST structures.

In order to evaluate the results of our research, we have studied the impact of narrative-based writing on a number of technical documents including presentations and websites. We also conducted an experiment involving a group of volunteers to test the ideas of a DN and the tool. These preliminary investigations have been encouraging.

1.3 A list of original contributions

The original contributions made by this research are listed below. They have been divided into primary and secondary contributions.

- **(PRIMARY) A new technique called narrative-based writing**

  A technique called narrative-based writing has been introduced (Chapter 4) which helps authors, particularly those working together in a team, develop a consistent story for their document. This technique brings together the key ideas from previously parallel strands of research: technical writing and narratives. Narrative-based writing was developed to address the shortcomings of other techniques, such as outlining, by providing authors with a way of working out the natural sequence of sections for their document. The use of RST in this context (i.e. to *synthesise* technical documents) differs greatly to its mainly analytical applications.

- **(PRIMARY) The design for a narrative-based tool for collaborative writers**

  The design of any collaborative working tool needs to be thorough. Our implementation is made more complex by the non-trivial RST tree structures (RS-trees) that need to be stored and maintained. Therefore, a combination of three models has been used to design the tool (Chapter 5): a conceptual model, a business process model and a functional model. The design culminates in a set of functions that are needed to manipulate the tree structures. This design furthers the understanding of narrative-based writing and modelling RS-trees. This design also allows the narrative-based functionality to be added, in the future, to existing collaborative working tools.

- **(PRIMARY) A web-based tool for collaborative narrative-based writing**

  A Web-based tool that implements the design is presented in Chapter 6. The tool was developed using Java, a JSP-driven HTML interface and a relational database. Alternative technologies such as XML were experimented with in two prototypes prior to this tool. This tool is a proof of a concept of the design.

- **(SECONDARY) A tutorial and a catalogue of case studies**

  We have produced a tutorial of narrative-based writing which is continuously being expanded. Furthermore, a set of generic DNs for popular types of documents such as a research paper, research proposal and presentation have been developed. Some of these DNs and their RST analyses are presented in this thesis as case studies in Chapter 7. The tutorial and the case studies will benefit technical authors when composing such documents.

- **(SECONDARY) Evaluation via an experiment and critical appraisal**

  The narrative-based technique and tool have been evaluated by us and also by a group of volunteers in an experiment conducted in May 2006. The outcomes of the experiment were encouraging and are described in Chapter 8.

1.4 Outline of the thesis

This introduction has, so far, outlined the problem addressed by our research, our solution and the main contributions made by our work. The rest of the chapters are organised as follows:

Chapter 2, *Background Literature,* defines document coherence and describes why collaborative writing makes coherence harder to achieve. We outline three methods by which authors can plan the structure of their documents and several collaborative writing tools. We show that they do not support document coherence. We discuss the connection between document coherence and narratives, and investigate possible discourse theories that could be applied to technical writing.

Chapter 3, *Rhetorical Structure Theory (RST)*, RST is the theory that was chosen for narrative-based writing. This chapter explains RST in detail and shows, by example, how it can be applied to a text.

Chapter 4, *Narrative-based writing,* introduces and explains the new technique.

Chapter 5, *A narrative-based collaborative writing tool: The design,* presents a detailed design for the tool. Three models are used for this design: a conceptual model, a business process model and a functional model.

Chapter 6, *A narrative-based collaborative writing tool: An implementation,* describes an implementation of the tool that is a proof of concept of the design in Chapter 5.

Chapter 7, *Case Studies,* contains case studies showing how narrative-based writing is applicable to various genres of technical communication, with particular benefits in collaborative writing.

Chapter 8, *Evaluation,* details the experiment conducted in May 2006 to evaluate our technique and tool. Then we provide a critical appraisal of our work and compare it to some related technologies.

Chapter 9, *Conclusions and future work,* presents the conclusions and areas of future work.

1.4.1 A document narrative for each chapter

To further illustrate the use of narrative-based writing, a DN has been included for each chapter in this thesis. The current chapter is an exception because it contains the DN for the entire thesis (Figure 1-2). Each DN was created after ruminating on the material intended for the chapter and formulating the most appropriate story. The DN is always given at the end of the chapter.

All DNs in the thesis will appear in text boxes with grey dashed lines as shown below. The RST analyses for some of them are included in the body of thesis. The rest can be found in Appendix A (section A.2). Most of the RST tree diagrams in this thesis have been drawn using the free software called RSTTool (O'Donnell, 2000).

The DN below summarises the story we want to convey via this thesis. Note that it has been divided into nine segments, each corresponding to one of the chapters.

[We believe that a narrative-based approach can help technical authors improve the coherence of documents they produce collaboratively.][1] [Coherence can be attributed to the story conveyed by a document. It is particularly difficult to get right in collaborative technical writing. Current writing tools do not support document coherence.][2] [Narrative and discourse theories, in particular RST, provide a solution.][3] [By combining the knowledge of these two parallel strands of research (narratives and technical writing), we have developed a new method of document structuring called narrative-based writing.][4] [In order to facilitate teams of geographically-dispersed authors to engage in narrative-based writing, we have carefully designed a tool][5] [and done a Web-based implementation of it.][6] [The new technique and tool are particularly beneficial in collaborative writing and can also be applied to other genres of technical communication such as websites and presentations.][7] [Preliminary investigations suggest that the narrative-based approach is helpful][8] [and that the tool, with some enhancements, can be a valuable contribution to technical authors.][9]

Figure 1-2: DN for the thesis

Chapter 2

Background literature

In chapter 1 we introduced the problem addressed in this thesis and outlined our solution. The aims of this chapter are to form a firm basis for understanding this problem further and to begin paving the path for the narrative-based solution that we propose in the forthcoming chapters. In order to do this, the chapter is divided into two parts.

Part I looks at what it means for a document to be coherent and why it is common for technical documentation to be particularly incoherent. One reason for this incoherence in technical documents is that they are often produced by multiple authors. We proceed, therefore, to examine the ways in which authors collaborate, highlighting the factors that lead to poorly structured documents. Finally, we present an overview of three current techniques that can be used to plan the structures of documents and some tools that help authors collaborate with their peers. We show that there is a clear gap in these areas with regards to document coherence.

Part II introduces the prospect of approaching document coherence from a narratives perspective. Previous texts have referred to the presence of an underlying narrative or story in a good document. The formal use of narratives in technical documents as shown in this thesis is a novel approach that combines previously parallel strands of research and will be explained in detail in chapter 4. We claim that the quality of a document can be improved by thinking of a better narrative for it. In preparation for chapter 4, part II of this chapter defines what a narrative is and examines some narrative theories (formally called "discourse theories") that have been developed by linguists and experts in narratology to analyse and synthesise better narratives. This is continued in Chapter 3 where the discourse theory chosen for our research, Rhetorical Structure Theory (RST), is discussed at length. RST is to become the mechanism by which we assess a narrative for a given document before writing any text.

The two parts of the chapter are presented below.

Coherence and collaborative technical writing

2.1 Document coherence: An informal definition

The concept of coherence is subjective and a precise definition is almost impossible. The use of language, the reader's prior knowledge of the subject area and even the layout of the text can all affect how coherent a document is to a reader. However, for the purposes of this thesis, it is necessary to discuss, and if possible specify, what is meant by the word 'coherence' within the context of our research.

Let us take the case of a set of related sentences, each of which is constructed well. If they are arranged haphazardly to produce a paragraph, it is unlikely that the paragraph will make much sense. Worse still, the paragraph may *appear* to convey a message but burden the reader with having to make non-existent logical connections between the adjacent sentences. With just a little bit of planning, the sentences can be positioned such that there is a natural, smooth progression of ideas between them making it easier for the reader to understand the paragraph just as the author intended it to be understood. It can even be said that such a paragraph conveys a consistent *story* or *narrative* to the reader. This aspect of a text is what we call **coherence**.

The above situation is best illustrated using the example below that has been taken from Alistair Knott's PhD thesis (1996). The figure shows two texts constructed using nearly identical sentences placed in different orders. The text on the left is coherent because it is easy to understand and conveys a story to the reader. The text on the right is incoherent. It is difficult to decipher and there are no obvious relationships between adjacent sentences.

The World in 1993

1993 will start with the world in a pessimistic frame of mind. That gloom should soon dispel itself. A clear economic recovery is under way. Though it will be hesitant at first, it will last the longer for being so. If you are sitting in one of the world's blackspots, this prediction will seem hopelessly optimistic. But next year's wealth won't return to yesteryear's winners; these middle-aged rich people need to look over their shoulders to the younger world that is closing in on them.

The World in 1993

1993 will start with the world in a pessimistic frame of mind. A clear economic recovery is under way. That gloom should soon dispel itself. These middle-aged rich people need to look over their shoulders to the younger world that is closing in on them. But next year's wealth won't return to yesteryear's winners; it will last the longer for being so if you are sitting in one of the world's blackspots. Though it will be hesitant at first, this prediction will seem hopelessly optimistic.

Figure 2-1: Example of a coherent (left) and incoherent text (right). Source: (Knott, 1996)

This definition of coherence can be extended to whole documents, both at the level of the sentences (like in the example above) and the level of sections or chapters. Similar thought

processes are necessary to work out the best order of the sections and how they would be linked together. This sort of planning for short texts like a paragraph may seem trivial. Many of us do it all the time in our conversations, e-mails and so on without paying much attention to it. However, for larger texts such as papers, theses or books, it is not so straightforward.

For instance, this chapter needed to explore a multitude of ideas and areas of research. Several plans were made to determine the best possible way of arranging all the sections or, to put it in another way, to determine the best possible story that this chapter could tell its readers. One might argue that the current version of this chapter is not appropriate either. However, the point being made is that it is not always easy to plan for and ensure coherence in a text, particularly in large documents (large *technical* documents, to be more precise).

2.2 Why focus on technical documents

By **technical documents**, we refer to everything from research papers and theses to conference presentations and websites. Technical documents, unfortunately, have a reputation for being poorly designed and difficult to read. Sometimes this is due to their scientific content being pitched at a level that is either too high or too low for the reader. Most times, though, the problems with these documents are related to coherence and how the information is pieced together.

- One explanation for this is that technical documents are often not written by people with formal training in writing or from linguistic backgrounds (Kieras, 1989). Winograd (1999) describes technical documentation as "that burdensome chore that managers are always trying to force onto recalcitrant…programmers". This may not be applicable to academic technical writing but rings true for some industrial settings. All this makes it harder for some technical authors to recognise and correct problematic texts. Furthermore, the tight deadlines to which these documents are produced mean that there is not much time to fix problematic texts.

- Another, more likely, explanation for the lack of coherence in technical documents is collaboration. It is very common to work with colleagues, in the same department or in different countries, to produce a technical document (e.g. a research paper). It was mentioned earlier that planning a coherent document is difficult. This is multiplied several times in **collaborative writing.** Imagine a scenario where many authors are contributing the sections of a document and also sharing opinions about where these sections should be placed. How would such a writing team arrive at the best possible story for their document? How would they make sure that each individual contribution adhered to this story?

For these reasons, the domain of our research is **collaborative technical writing.** Technical writing was chosen for two other reasons too. Being computer scientists, most of our writing experience so far has been in this area, making it a suitable genre to apply our research to.

Also, later in the thesis, we recommend generic structures for types of documents such as research proposals. This cannot be done with other, less structured types of writing such as creative writing. Even though there may be anticipated formats for some creative texts (e.g. the typical set of moves in Figure 2-12 that is expected in a James Bond novel), others may deliberately be constructed to defy recommended structures for added effect (e.g. a novel with an unexpected twist at the end or a poem). We focus, therefore, on technical documents produced collaboratively.

2.3 Collaborative writing

Collaborative writing (CW) is the process in which multiple authors work together to produce one document. It is not just the soliciting of ideas about the document but the actual contribution of the various sections which are then collated together to form the final document.

CW has several advantages over single-author writing. In a survey done by Noël and Robert (2004), the participants agreed that CW resulted in richer documents owing to diverse ideas, input from co-authors with different expertise and task distribution. Theoretically, CW should also be more efficient. Each author would have to produce just a section instead of the whole document; when done in parallel, this should save time. Assuming that each section is written by the relevant expert in the team, the sections are likely to be better and more accurate as well.

The disadvantages of CW include difficult group management and coordination (Noël and Robert, 2004), and documents that are poorly structured. Extra coordination is needed in CW, especially when the authors are geographically dispersed. The sections contributed by the authors may need to be edited to fit the eventual document structure. All this could lead to an *increase* in the time spent, in comparison to the time required for a single author to write the same document. The final document may also have some problems with coherence. For instance, some authors lower down in the hierarchy may not be aware of the whole purpose and structure of the document (e.g. a PhD student delegated some writing by his supervisor). The sections thus created may not fit together properly leading to documents that have been described as ‘arbitrary’ (Lowry et al., 2004).

Nevertheless, CW is becoming increasingly popular and there are, broadly, two ways in which these authors can opt to work (see below). Each model has its own pros and cons with regards to the quality of the eventual document.

2.3.1 Ways in which authors collaborate

When authors work collaboratively, they can choose to coordinate their work in one of several ways. All these methods can be divided into two models: sequential and parallel.

Sequential writing model

In this model, only one author can edit the document at a given time and once his/her task is complete, passes the document along to the author next in the chain.

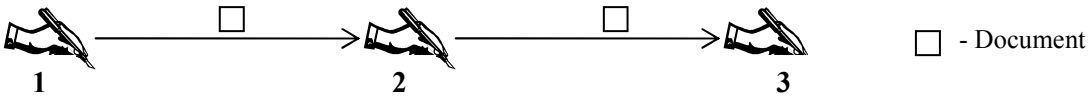


Figure 2-2: Sequential writing model

This model is easy to organise and improves coordination between the authors. Each author can read the previous authors' work before making his contribution. This can help improve coherence. However, there are some disadvantages (Lowry et al., 2004) in this model such as the lack of group consensus and the difficulty in ensuring that all document sections are addressed adequately. Unless the team reviews the document, there is no way of finding out if all the sections meet everyone's expectations and fit the story that was intended for the document. Also, the *order* of the authors greatly influences the final document. One author can change previous contributions or bias subsequent authors.

Parallel writing models

In this model, a team divides the writing task into discrete units and works in parallel. This improves group consensus and efficiency. If the authors are able to view the rest of the document as they write, it is more beneficial for document coherence.

There are several variants of this process. In one, team members are assigned roles depending on their expertise such as 'writer', 'reviewer' and 'editor'. Members then work on the document according to their roles. In another variation, the document is divided into sections

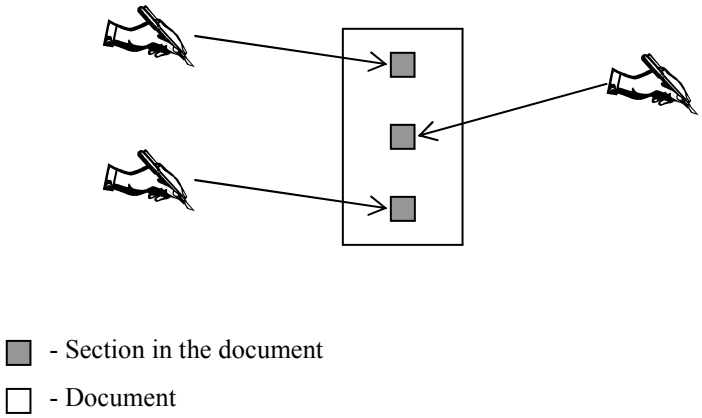


Figure 2-3: Parallel writing model

and each author is assigned a section that he/she is responsible for. The completed sections are submitted to the team leader who assembles them together to form the final document. This approach is sometimes called *horizontal-division writing* (Lowry et al., 2004) and can, unfortunately, result in arbitrary sections that do not constitute a coherent story in the eventual document. Alternatively, the team leader will have the mammoth task of editing each section to make them fit. In (Alred et al., 2003), the sequence of steps in this parallel writing process is listed as:

1. Designate one person as the team coordinator.
2. Collectively identify the audience, purpose and project scope.
3. Create a working outline of the document.
4. Assign segments or tasks to each team member.
5. Establish a schedule: due dates for drafts, revisions, and final documents.
6. Agree on a standard reference guide for style and format.
7. Research and write drafts of document segments.
8. Exchange segments for team member reviews.
9. Revise segments as needed.
10. Meet your established goals.

Both the writing models above have their advantages and also their weaknesses with respect to the time taken, workload on the team leader and, most worryingly, the coherence and consistency of the final document produced. One way of improving this situation is by planning the structure of the document at the start of the writing process (as depicted by step 3 in the list above).

2.4 Three techniques to organise the ideas in a document

There is evidence to suggest that a period of planning can significantly enhance the quality and coherence of a document (Torrance and Bouayad-Agha, 2001). Such a plan makes the author aware of the goals that the entire document (and individual sections) should fulfil and the structure it should adhere to. Outlines are a popular method of planning documents. In this section, we discuss outlines and two other planning techniques. We concentrate on the impact these methods have on the coherence of collaboratively written documents, traits that make them popular and gaps in their structuring methodologies.

2.4.1 Mind maps

Mind maps are diagrams that help authors strategically *visualise* their thinking on a particular topic. They are organised displays of information. Figure 2-4, for instance, shows the key ideas in this chapter. The method by which these diagrams are drawn is called mapping or, sometimes, clustering (Roth, 1999). Authors start with a topic at the centre and then generate a web of related ideas from that (Steele, 2002).

In a mind map, key ideas are usually enclosed in boxes (or "clouds") and lines (sometimes labelled) are used to indicate relationships between these ideas. Some authors also use various ways to differentiate the main concepts from the sub-concepts. In Figure 2-4, for instance, the levels of concepts are indicated, in order, using dark, dashed and normal lines in the boxes (and circles).

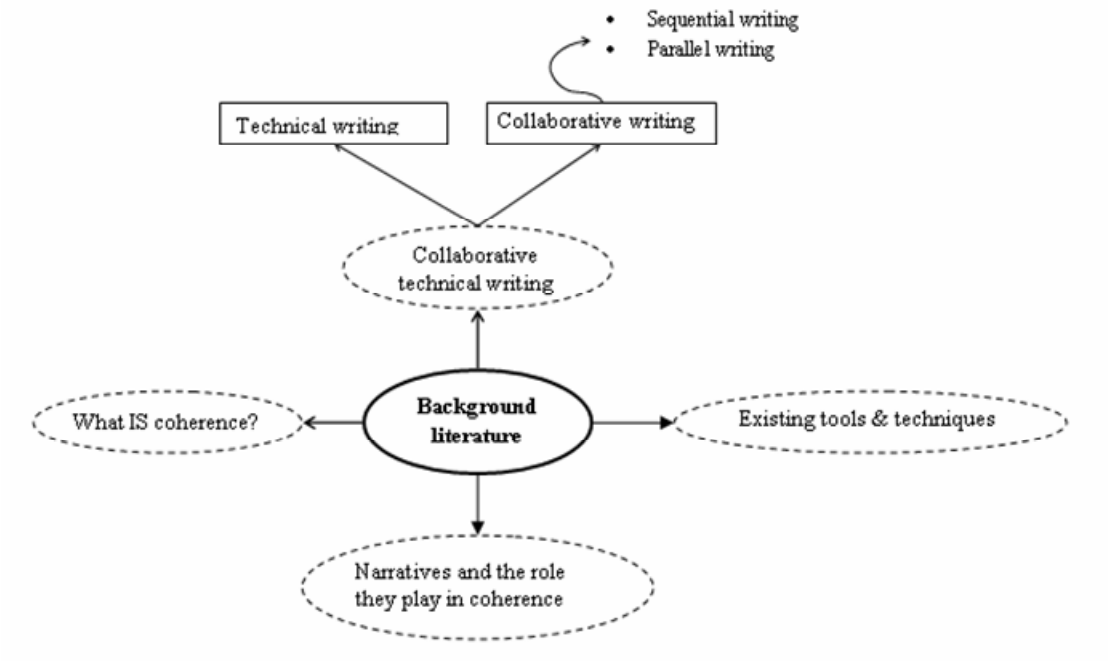


Figure 2-4: A map showing the ideas that are presented in this chapter

Mind maps are good visual aids. In writing, one can imagine a group of authors sitting in a room discussing a document and the team leader plotting the corresponding mind map on a white board. There can be debates about the inclusion, exclusion and importance of concepts and the how they relate to one another. At the end of the session, the authors have a logical picture or model of the document in their minds. This mental image will serve well when formulating the content of the document.

However, the benefits of using a mind map are unclear when the writers are far apart, having no or infrequent face-to-face meetings. Much of the information in a mind-map is gained by *understanding* the associations made between ideas. The questions or relationships raised by the connecting arrows may be ambiguous. So, how well would a mind map drawn by one author in the team communicate the ideas to a second author?

Furthermore, in order to become a practical guide to writing, a mind-map needs to be transformed into a linear format, such as an outline, to reflect the *order* in which the ideas will appear in the document. There is no definite way to derive this linear format from a mind map and, once again, may differ from author to author. Therefore, a mind map alone is not an adequate planning technique.

In summary, the diagrammatic representation of ideas in a mind map is useful but the lack of a defined process by which it translates into a linear structure for a document is frustrating and, possibly, detrimental to the document.

2.4.2 Outlines

Outlines are, by far, the most popular way of planning a document. An outline is "an orderly plan…showing the division of ideas and their arrangement in relation to one another" (Roth, 1999). An example of an outline is given below, showing the organisation of the content in this chapter.

Outlines can be composed of noun phrases (as shown in Figure 2-5) or whole sentences. Relationships between the phrases are shown by indentation (main topic and sub topics) and using the same kind of symbol for equally important ideas (A,B,C… I,II,III...1,2,3 and so on).

PART I
- 2.1 Document coherence
- 2.2 Why technical documents
- 2.3 Collaborative writing
  - *Sequential writing*
  - *Parallel writing*
- 2.4 Three techniques
  - *Mind maps*
  - *Outlines*
  - *Pyramids*
- 2.5 Tools

PART II
- 2.6 Narratives and theories

2.7 Summary of the chapter

Figure 2-5: Outline for this chapter

Outlines transform notes, drafts and other material into an ordered progression of ideas (Lester and James D. Lester, 2005). There is a direct correspondence between an outline and the sections in the document, making it a useful reference and guide.

A writing team can agree on an outline at the start. If the team is geographically dispersed, this is something that can be done by e-mail. Phrases can be inserted and deleted, and their order changed. The added advantage is that all the team members are likely to be able to relate to outlines since the technique is popular and easy.

However, outlines lack the explicit connections between the ideas that were made in mind maps. Even though hierarchical relationships are shown (such as sections, sub sections and so on), there is little indication of the *purpose* of each section and the role it is meant to play. This latter kind of information will help improve coherence since it advises authors on how to structure their text. For example, section 2.4 in the sample outline above is expected to discuss planning techniques. But *how* should this discussion be crafted? *Why* is this section in this chapter? What *effect* is it supposed to have on the reader such that it prepares the reader for the next section?

So, in summary, outlines are popular, easy to understand and have a one-to-one correspondence with the document. There is, however, a lack of information in them to support overall coherence. An outline only seems to function as scaffolding for the document.

2.4.3 Pyramids

Pyramids are another way of structuring information in a document. For years, journalists used "inverted pyramids" when writing newspaper articles. This meant that they placed the most important piece of information (often the conclusion) at the *start* of the article. Graphically, this was represented by the broadest part of the pyramid being at the top (hence, inverted). The rest of the article would contain information of *diminishing* importance. This had the advantage that if the reader left the article at any given point, he would still have the whole story (or certainly, the general gist of it). This was ideal for newspaper reports and web pages.

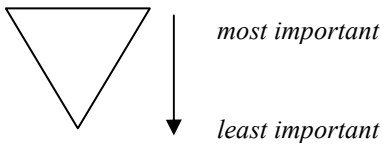


Figure 2-6: The inverted pyramid structure used by journalists

However, for most documents there needs to be an introduction at the start and a gradual build up to the most important part of the document (for example, theses and papers). This is often depicted by drawing the pyramid the "right way up" (i.e. with the broadest part of the pyramid at the bottom). The Pyramid Principle is a technique introduced by Barbara Minto (2002) that formalises the use of pyramids in this way. The important feature in this technique is the organisation of ideas such that it corresponds to the *reader's* thinking.

She recommends structuring the ideas in a top down fashion to form a pyramid as illustrated in Figure 2-7. The top-most box in the pyramid presents the central subject (the introduction) under which all the other ideas belong. The pyramid is governed by vertical and horizontal relationships.

The vertical relationships in the pyramid are 'question and answer' relationships. Each box in the pyramid is expected to raise certain questions in the mind of the reader. These questions should be answered in the boxes immediately below it. The questions raised by this next level in the pyramid should be answered in the level below. This is continued until the writer is confident that the reader will have no more questions. As a general rule for a good document, Minto advices not to answer a question before it has been raised in the reader's mind or to raise a question that will not be answered.

Each level in the pyramid also has horizontal relationships. These are logical relationships. The boxes should not only answer the questions raised in the line above them but answer them *logically.* This is done by presenting clearly either an *inductive* or *deductive* argument. To illustrate this process further and to compare it to the previous techniques, the pyramid principle has been applied to the structure of this chapter too (see Figure 2-7). More details about the Pyramid Principle can be found in (Minto, 2002).

An important point in this technique is the consideration of the document from the *reader's* perspective. The defined need for relationships between and across levels in the pyramid contributes to coherence. The technique leads to a diagrammatic display of ideas that provides a good visual aid. It also relates to the hierarchical structure of a document (main topics followed by sub topics underneath them) and the linear sequence of the sections.

The question-and-answer dialogue with the reader is also ideal, creating a certain amount of curiosity in the reader's mind before quickly supplying the answer. The Pyramid Principle contains the right amount of detail necessary to improve coherence. Focusing on creating these logical relationships in the text will no doubt lead to better documents.

The main disadvantage, in our opinion, is that the pyramid principle is comparatively harder.

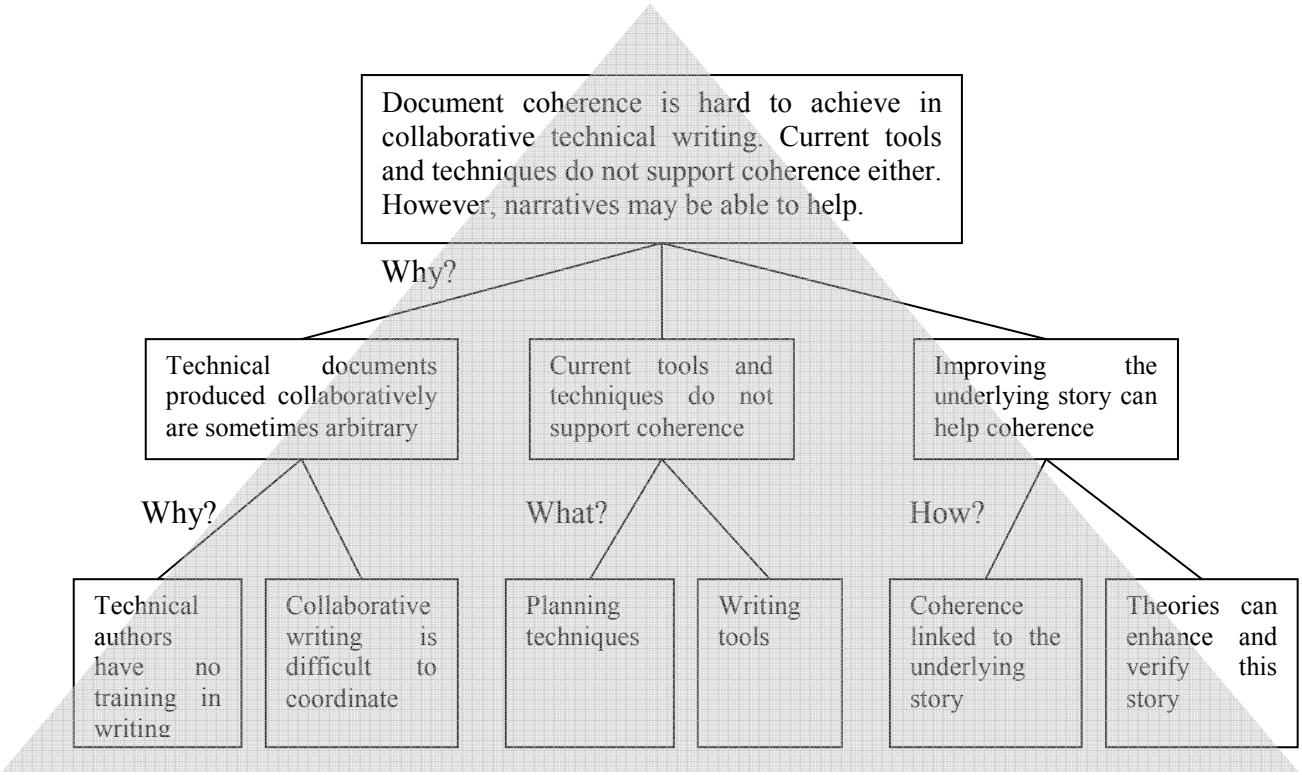


Figure 2-7: Barbara Minto's pyramid principle applied to the content of this chapter. There are vertical question-answer relationships and horizontal logical relationships in the pyramid.

To sum up: We have studied three techniques with regards to their ability in improving the coherence of co-authored documents.

- A graphical representation of ideas like a mind map is useful but cannot be relied on entirely. Sooner or later, a mind map has to be converted into a linear format to work out the sequence in which the ideas should appear in the document.
- Outlining is a popular and easy technique to derive the linear sequence of sections in a document. However, an outline does not have enough information to justify this linear format or advise the authors about the *logical* tone that their individual sections (and the whole document) should take.
- The pyramid principle gives a good, well-defined structure to create this logical flow in the document. While being useful, it is relatively complex.
- All three techniques also have no way of guaranteeing (or evaluating) coherence.

We recognise a need for a planning technique for co-authors that will be a **graphical model**, provide a **natural ordering of sections** and **connect these sections with logical relationships** that justify their existence and location in the document. With respect to collaborative writing, the technique also needs to be such that plans produced by one author can easily be transferred to and understood by another author. These criteria are re-examined in chapter 4 when we introduce narrative-based writing.

2.5 Collaborative writing (CW) tools

In section 2.3, the difficulties in achieving document coherence in CW were raised. We mentioned that one way that this situation can be helped is by using a planning technique prior to writing. Another way is to make use of an appropriate software tool. Having studied existing planning techniques in the previous sections, we now move on to look at some tools that support collaborative writing.

Writing software are many and varied (Palmquist, 2003, Porter, 2003), supporting various aspects of the authoring process. For instance, some tools aim to increase the productivity of the author and enhance the layout of a document, famously Microsoft Word and LaTex. Such tools are powerful in what they were designed to do but have little relevance to coherence or collaborative writing. The focus of this section is, therefore, on tools that aid collaborative writing. We consider these tools with respect to two questions:

1. How do these tools support collaboration? (to identify a set of features supported by a majority of collaborative writing tools)
2. Do these tools help improve coherence?

The idea of computer support for collaborative working has been around for several decades. It has even generated a dedicated field of research called CSCW (Computer-Supported Cooperative Work). It is impossible to explore *all* the software that has since been generated to support collaborative working. There are just too many of them. So, we examine only a demonstrative sample of collaborative writing tools, presented below in chronological order.

The first in our list is a tool called **Quilt** from 1988 (Fish et al., 1988). It allowed users to change, annotate and share documents. It provided messaging, computer conferencing and notification facilities to support communication between the collaborators of a document. Another stand-alone application of that era was the **PREP** editor (Neuwirth et al., 1992, Neuwirth et al., 1990, Neuwirth et al., 1994). Instead of storing whole documents, it introduced a concept called a ‘chunk’ (Neuwirth et al., 1990) which roughly corresponded to an idea and was able to contain text, grids or images. One of the important features of PREP was the use of a flexible difference-finding algorithm to find discrepancies between versions of documents. This enabled authors to see, at a glance, the changes that other authors had made to the document. There was no obvious support for document structures and coherence. These tools are presented here mainly for their historic value. They were developed at a time when CSCW was still in its infancy and, while demonstrating useful concepts such as version control and tracking changes, the success of both of them was limited.

There were several other collaborative tools developed at the time (Genthial and Courtin, 1992). However, with the success of the WWW, the use of web-based tools for CW was becoming increasingly popular. Even the standard HTTP/1.1 protocol was extended to support the features necessary for asynchronous CW. The new protocol was called **WebDAV** (World Wide Web Distributed Authoring and Versioning). It provided additional services for

editing and managing files on remote web servers in a secure way (e.g. locking, version management, access control and so on) forming a good basis for building web-based CW applications (Dridi and Neumann, 1999). Today, there are many web-based tools that assist collaboration. We discuss a couple of them.

Writely[1] is a free web-based tool that allows authors to upload documents (in one of several formats) and collaborate with specified co-authors in real time. Documents can be edited online via an interface similar to that of Microsoft Word. If someone else is working on the same document at the same time, the changes are merged instantly. There is offsite storage and backups are made every ten seconds. The revisions of the documents can be compared and authors can roll back to previous versions (thus 'undo'ing some updates). Users accessing the document can be owners, collaborators or viewers (user roles). Owners own the document and can edit and delete it. Collaborators can edit the document and invite other collaborators. Viewers can only read the latest version of the document but cannot make changes.

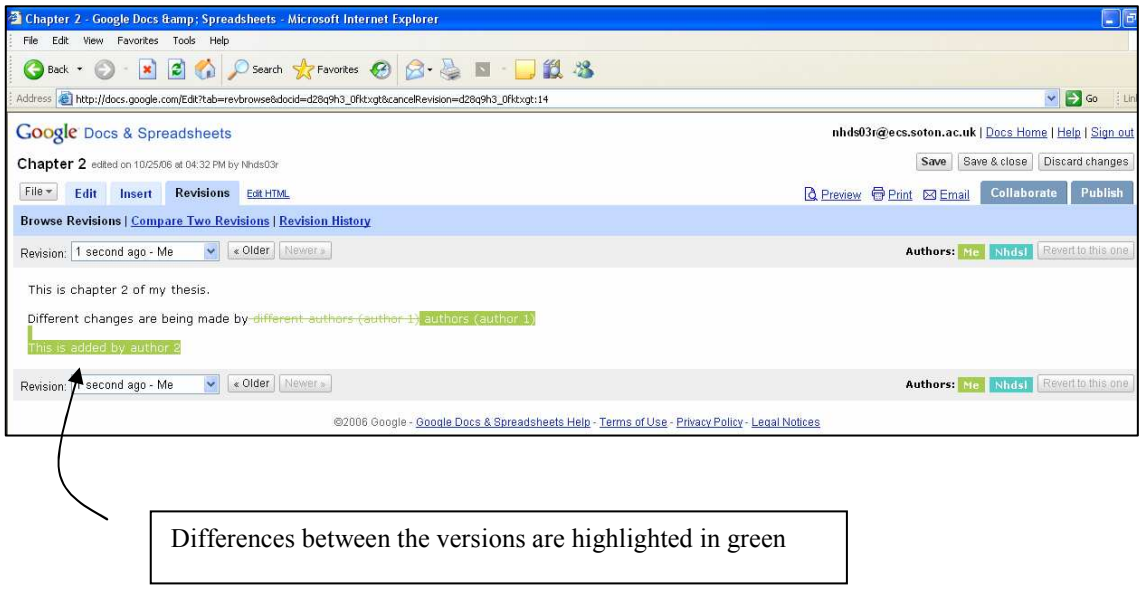


Figure 2-8: An example of a document in writely. The figure shows two versions of the document being compared.

Writely possesses all the basic requirements for CW: version control, tracking changes, authorising users and so on. Furthermore, web-based tools are contemporary, are simpler to build and allow easy access. To use Writely, for example, users only require a web browser.

We conclude the discussion on web-based tools with a quick look at **Wikis.** Wikis are websites that allow its users to add, edit and remove the site's content. They are among the newest forms of communication on the web. A history of revisions is often maintained and therefore, changes can be undone by reverting to an older version. The uniqueness of Wikis is

[1] http://www.writely.com/

that there is no prior determination of the users that are allowed to edit the content and is, in this sense, open to all. This can have both advantages and disadvantages. **Wikipedia**, for example, is a thriving wiki. Readers contribute the facts with some minimal supervision from the host site. The varying expertise of the readers means that the information is thorough and up to date.

At the same time, there is always a question about the validity of the information on wikis (Johnson, 2006). There is no guarantee that individuals contributing to wikis are trustworthy and unbiased. An example where a wiki failed was the first ever **wikitorial** hosted by the LA Times in June 2005. Their editorial was re-written by about 400 "wikipedians". However, two days after its launch the site had to be taken down because of some inappropriate content that was posted online.

Version management software like **CVS** and **LibreSource** also assist collaborative writing by maintaining a systematic record of revisions and merging versions (see section 5.2 for a discussion of these tools with regards to version control). Merging is an important point to raise here. Co-authors working on individual local copies of the document are bound to make conflicting changes. Several merge algorithms have been developed to integrate these changes. A classic approach is to reproduce every change in all the copies of the document. However, with this method, the merged result may not always be correct.

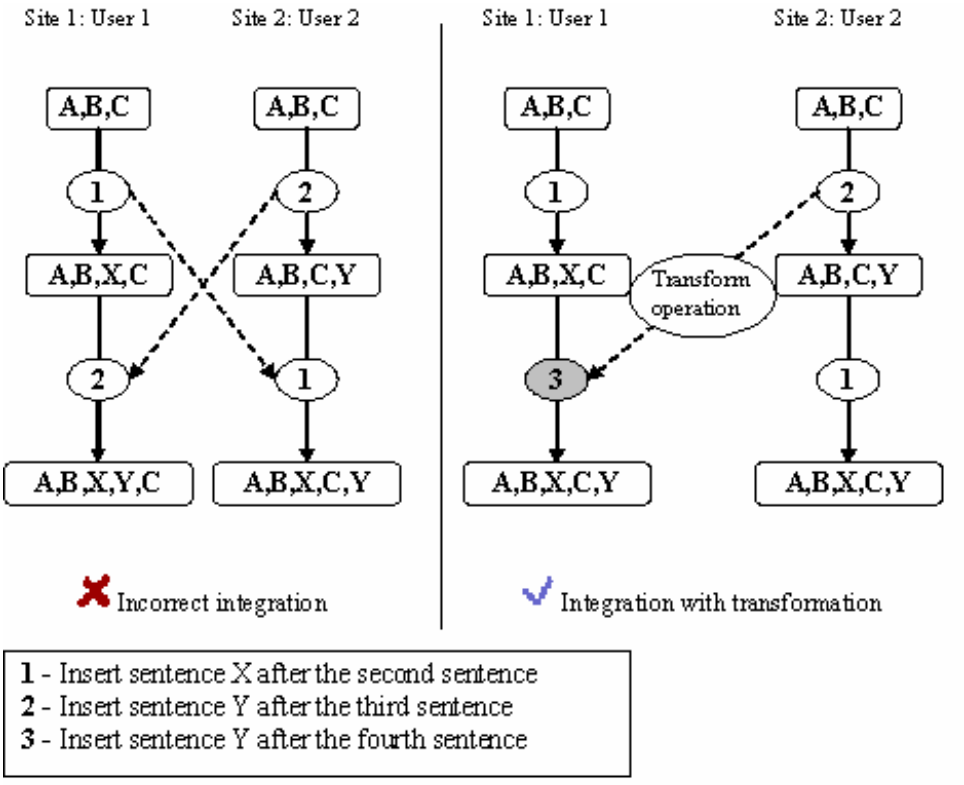


Figure 2-9: Integration of changes using the Operational Transformation method (Source: (Molli et al., 2003))

Figure 2-9[2] shows two authors editing a paragraph which initially has three sentences labelled A, B and C. Both authors insert new sentences. On the left, the change made by one author is just replicated in the other author's copy. This results in two different merged versions (ABXYC and ABXCY).

A merging technique called Operational Transformation (used in LibreSource) (Molli et al., 2003) improves this situation by integrating the changes made to the individual copies *and* preserving the context in which they were made (referred to as 'intention preservation'). So, on the right hand side of Figure 2-9, the operation performed by User 2 has been modified so that it takes into account the previous operation done by User 1. This results in both users having the identical versions of the text (ABXCY). This algorithm provides some assistance to coherence by making sure that the merged versions have all the changes integrated correctly. However, syntactic equivalence alone does not guarantee coherence and is not a complete solution to our problem (De-Silva and Skaf-Molli, 2006).

Our study into writing tools also included software that provided templates for users to create documents. For instance, the **Newnovelist**[3] software (developed and published by Creativity Software Ltd.) claims to help a novice write a novel in five steps. The tool provides fixed templates for various genres of novels. However, these tools were not included here because they give no leeway to authors to be creative or explore document structures on their own. These are not feasible solutions to improving coherence of documents even though the suggested templates can be good guidelines to the writing.

One may also notice the absence of tools like **Mindmapper**[4] - that helps draw mind maps – and tools that assist in the creation of outlines in our discussion. This is deliberate because it was decided earlier (section 2.4) that mind maps and outlines were not sufficient to improve the coherence of documents. Also, the number of writing tools is far too many for us to be able to cover everything here.

2.5.1 Discussion

Writing tools have had varied success in appealing to collaborative writers. Some authors only ever use e-mail to communicate their ideas and word processors to produce the documents that they then mail to each other (Noël and Robert, 2004). However, despite these debates about the usefulness of specialised collaborative writing tools (Noël and Robert, 2004, Pargman, 2003), programmes like CVS and even the "track changes" option in Word are in popular use.

Software tools do help collaborative writing in many aspects. Some useful features of collaborative writing tools are listed below:

---

[2] This diagram is similar to the figure in (Molli et al, 2003). We focus on changes to a paragraph while their figure illustrated changes to a single word.
[3] Available from www.amazon.co.uk
[4] http://www.mindmapper.com/

- Being able to track changes
- Version control and the ability to revert to previous versions
- Merging of versions (and thus recognising syntactic conflicts)
- Being able to identify the contributor of each change
- Controlling who has rights to access and change the document
- Some tools are web-based allowing easier access

The area of collaborative writing that our research focuses on is coherence. An explanation of what we mean by the word coherence was given in section 2.1. We were, therefore, looking for software that supported co-authors make decisions about which section should go where in a document so that it presented a good story. However, we were unable to find such support in the tools that we have come across during the course of our research. The closest the tools got to improving coherence was by making sure that the authors worked on consistent copies of the document and by preserving the context in which changes were made to these copies. This, however, is no guarantee of coherence.

This prompted us to look into ways that coherence could be better supported.

Narratives and narrative theories

2.6 The role of narratives in technical writing

A **narrative** can be broadly defined as "the…representation of a series of events meaningfully connected in a temporal and causal way" (Onega and Landa, 1996). While some researchers distinguish between a story and a narrative (Lothe, 2000, Abbott, 2002), others use the two words interchangeably. In this thesis, a narrative is considered to be analogous to a story.

The word 'narrative' has been used in connection to the structuring of technical documents in several texts. For instance, when describing the pyramid principle, Barbara Minto (2002) uses the terms "narrative flow" to emphasise the need for a smooth story in the introduction. Evans and Gruba (2004) say that a thesis should "read like a novel", thus implying the need for a story or a smooth progression of ideas. Similarly, other researchers such as Zobel (2004) have alluded to the need for a story (or narrative) to improve documents in computer science. We too turned to narratives in our attempt to address coherence. It was a natural development of our research.

The need for a consistent narrative is apparent, in our opinion, in novels, movies and other stories. A murder mystery, for example, will only succeed if the plot unravels in a logical sequence. This requirement is less obvious in technical and business writing where the emphasis is mainly on the factual content as opposed to the *storyline*. However, even technical documents are more effective if they present a well planned story to the reader. One of the major faults with technical writing, particularly ones with multiple authors, is that the various sections do not seem to fit together properly. We, therefore, attribute the coherence of a document to the narrative it conveys to the reader. We claim that the more coherent this narrative is, the more coherent the document will be. This idea is the basis for the new technique we introduce in chapter 4.

Having identified the role that narratives can play in technical documentation, we went further to explore methods by which narratives could be improved. Were there defined ways to structure better narratives? Theories? It turned out that there were many theories developed by linguists and experts in narratology to analyse and synthesise well structured narratives. We realised that using such a theory will benefit technical documents since the underlying story in them could be verified and improved. With this intention, we discuss some of these theories below.

2.6.1 Understanding and improving narratives

It appears that there are two ways in which narratives have been studied. The first was to identify regular structures for a genre of narratives and use this as a guideline when creating

new text belonging to this genre. The second was to develop mechanisms by which the human thinking process behind the construction of a successful narrative could be modelled and emulated. A similar distinction is made in (Lang, 1999). We discuss both these approaches, presenting three theories for each.

For the first category, we discuss the work of Gustav Freytag, Vladimir Propp and Umberto Eco. They have identified regular structures for a play, the Russian folktale and James Bond stories respectively.

(1) In 1863, the German journalist and writer, **Gustav Freytag**, recommended a five-part pyramidal structure that he believed was the most successful format for a play (Freytag, 1863).

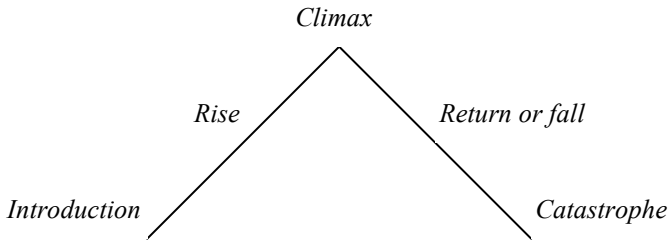


Figure 2-10: Freytag's pyramidal structure for a play (Freytag, 1863)

Each part in Freytag's pyramid has a specific function and could consist of a single scene or a succession of connected scenes (an act). As the name suggests, the *introduction* introduces major characters, sets the theme for the plot and perhaps even hints of the forthcoming conflict (Wheeler, 2004). The second part, *rise,* represents an increase in tension or uncertainty. *Climax* is the part with the greatest tension and audience involvement, usually situated in the third act of the play. The fourth part usually has falling fortunes for either the hero (if it is a tragedy) or the antagonist (if it is a comedy). This section culminates in the final part called the *catastrophe*. If it is a tragedy, the final part will be a disaster involving the hero and his loved ones. If it is a comedy, this will be a denouement (or conclusion) leaving the antagonist worse than he started off with at the beginning of the play. After this final disaster, the suspense and tension ends, providing the audience with closure.

(2) Similarly, **Vladimir Propp** studied 128 Russian folktales (Propp, 1928) and arrived at a sequence of 31 generic *narratemes* (smallest narrative units), most of which were present in each folktale. A few of these narratemes are listed below. Propp's narratemes have been implemented in software such as the `Joseph` system (Beaubouef and Lang, 1998) to generate tales that resembled folktales.

1. A member of a family leaves home (the hero is introduced);
…
12. Hero is tested, interrogated, attacked etc, preparing the way for his/her receiving magical agent or helper (donor);
…
16. Hero and villain join in direct combat;
…
30. Villain is punished;
31. Hero marries and ascends the throne (is rewarded/promoted).

Figure 2-11: A few of Propp's narratemes for the Russian folktale

(3) The third example we present in this category is the classic sequence of moves identified by **Umberto Eco** (1979) in all of Fleming's James Bond novels. In each story, Bond's 'boss' (M) assigns him a task which evokes some opposition from a villain. There is usually a woman who becomes involved in the conflict. The villain has temporary success when he captures Bond. However, in the end, the villain is always defeated and James Bond emerges as the winner. Below, this structure is repeated verbatim from (Eco, 1979).

A. M moves and gives a task to Bond;
B. Villain moves and appears to Bond (perhaps in vicarious forms);
C. Bond moves and gives a first check to Villain or the Villain gives first check to Bond;
D. Woman moves and shows herself to Bond;
E. Bond takes Woman (possesses her or begins her seduction);
F. Villain captures Bond (with or without woman, or at different moments);
G. Villain tortures Bond (with or without woman);
H. Bond beats Villain (kills him, or kills his representative or helps at their killing);
I. Bond, convalescing, enjoys Woman, whom he then loses.

Figure 2-12: Scheme of moves for a James Bond novel. .M represents James' 'boss' (Copied literally from (Eco, 1979))

We presented three regular patterns identified for specific types of stories and plays. For the most part, readers welcome new but familiar patterns (Sharples, 1996). Take the James Bond movies for example. The audience is aware that James Bond will always win in the end but still enjoy watching the different settings and twists that make it difficult for Bond to win.

This latter point is also the flaw in these fixed templates. For creative writing, generic structures do not do justice to the slight variations to the patterns that make each story unique. For technical writing, this is not so much of a problem since most documents have similar patterns. However, these templates do not give authors an *understanding* as to why they are more popular than others. Such knowledge will help authors make decisions about whether or not their documents are good and coherent.

The second approach to narrative analysis has led to theories that, in our opinion, equip the authors with knowledge of how stories work. This allows them to create, analyse and justify suitable structures themselves. This is more suited to enhancing the coherence of technical documents than just providing authors with templates. We looked at several of these theories with the aim of picking one for our research. We discuss three of them below with regards to the following criteria:

1. Simplicity
2. Complete definitions that help the authors *produce* texts as opposed to just analysing them
3. Ability to provide some way of judging if a narrative is coherent
4. Suitability for technical documents

In order to illustrate each of the theories better, we apply each one to the simple, light-hearted example below.

> Fido is usually a happy dog.
> One day, Fido was unhappy because he had got fleas and could not stop scratching.
> The vet recommended a flea treatment which got rid of the fleas.
> Fido stopped scratching and was happy again.

(1) First, we present Wendy Lehnert's theory (Lehnert, 1981). This theory is a bottom up approach to analysing texts by breaking them up into **affect states**. An affect state is a smaller entity that occurs with respect to one character. There are three types of affect states.

+	Events that please (positive events)
-	Events that displease (negative events)
M	Mental states of the character (neutral effect)

A map of chronologically ordered affect states can be drawn for each character of the story (see Figure 2-14). Diagonal links signify causalities of affect across characters.

Lehnert recognised about fifteen recurring patterns in which these affect states occurred. She called these groups of affect states *plot units*. A story could be summarised by using its plot units. Two examples of plot units are given below. The success plot unit depicts a mental state of a character leading to a positive event. The failure plot unit leads to a negative event.

Figure 2-13: Examples of two plot units identified by Wendy Lehnert

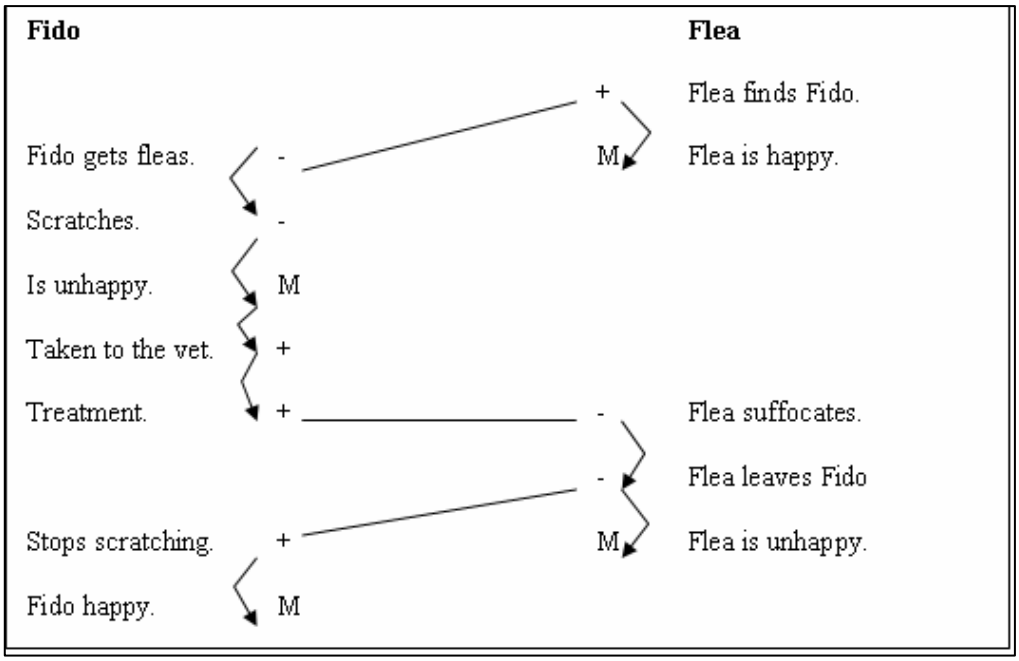


Figure 2-14: Wendy Lehnert's affect states applied to our short story

This is a reasonably simple theory to understand but while been useful to *analyse* texts, it does not provide much guidance regarding the *production* of texts. Ideally we want a theory that will help authors generate better narratives. Also, the reliance on characters in the analysis is not appropriate for technical writing.

(2) The second theory that we found interesting is the **classification of narrative events** by Bremond (1980). Narrative events in a story were divided into two basic types: "amelioration to obtain" and "degradation expected". He describes three ways in which these two types of events can be combined in a story. One such combination is called *coupling*. This is when the amelioration of the fate of one character coincides with the degradation of the fate of another character (with opposite interests) like in our sample story (see below).

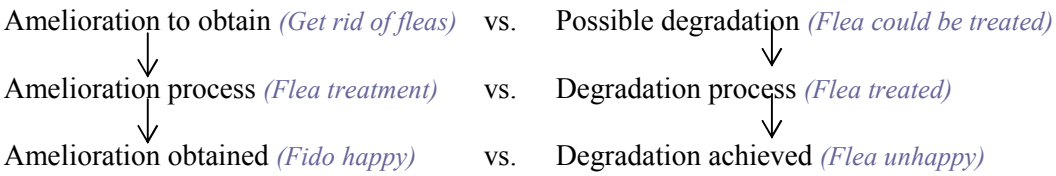


Figure 2-15: Applying Bremond's theory to our sample story

Once again such a theory would not be applicable to technical writing. It was interesting because it presented ways in which events could be combined to form different types of stories and we were curious to find out if this approach was possible in technical documents too. The answer, in our opinion, is no.

(3) Finally, we present **Centering Theory** (Grosz and Sidner, 1986, Grosz et al., 1995). This theory focuses on the *local coherence* of a discourse. A discourse is a sequence of utterances that exhibit coherence. The theory suggests that each of these utterances has a *centre* which links it to other utterances in the discourse. Each utterance is assigned a set of *forward-looking centres* and a *single backward-looking centre* (except the first utterance in the discourse). The symbols for these concepts are given below.

C – Centre
U – Utterance
$C_f(U_n)$ – Set of forward looking centres of utterance U_n
$C_b(U_n)$ – The backward looking centre of utterance U_n

The eventual coherence of the text is determined by the nature of the transition of these centres from one utterance to the next. They define three ways in which this transition can happen. We examine one of them below.

Centre Continuation:
$C_b(U_{n+1}) = C_b(U_n)$ AND this entity is the most highly ranked element of $C_f(U_{n+1})$. In a coherent discourse, $C_b(U_{n+1})$ will go on to become $C_b(U_{n+2})$ and so on.

This is the easiest transition for the reader to understand and is shown below.

U_1 : Fido is usually happy.
U_2 : He suddenly became unhappy and started scratching.
U_3 : He was taken to the vet.

The backward and forward centre of these utterances is Fido.

The two other transition types are Centre Retaining and Centre Shifting **(**$C_b(U_{n+1}) \neq C_b(U_n)$**)**. Center shifting is demonstrated in the example below and it is clear why a reader would find it difficult to comprehend.

U_1 : Fido had fleas.
U_2 : He was taken to the vet.
U_3 : He is generally very good with animals.

U_3 is confusing. It actually refers to the vet but it could easily be taken to be about Fido.

The theory goes on to present two rules that govern the centre transitions of coherent discourses and is therefore able to guide an author towards a better narrative. According to the theory, an author should aim for centre continuation and avoid centre shifts. The examples above are all of very short texts but this theory can, in our opinion, be extended to whole documents. In this case, the rules will apply to the centres of entire *sections.* Due to the formal nature of its definitions, this theory appears to be something that can be implemented in

software. The drawback of this theory is that it is time consuming to apply and has no indication of the logical relationships between utterances.

Centre continuation is definitely important for coherence. However, is it the *only* requirement for coherence? Are there other factors that govern the quality of the underlying story told in documents? The answer came in the form of **coherence relationships**; the concept of there being interdependencies between parts of a text that went beyond just centre continuation.

Several researchers pointed out that there were implicit relationships between segments of a text that held it all together and made it more coherent. One such researcher was J. R. Hobbs who introduced a set of coherence relations (Hobbs, 1982, Hobbs, 1985):

Occasion	*Elaboration*
Evaluation	*Exemplification*
Background	*Contrast*
Explanation	*Violated expectation*
Parallel	

For instance, in our sample story, the fact that Fido is usually a happy dog provides background information to the events that follow next. Without it, the observation that Fido is unhappy bears no special significance.

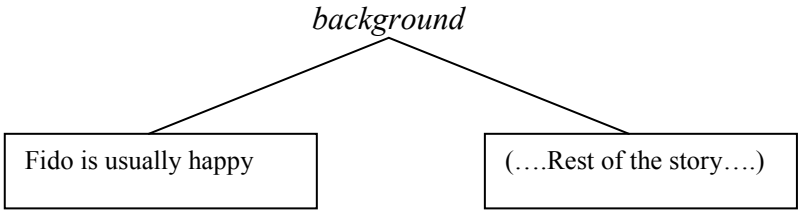


Figure 2-16: A coherence relationship

The rest of the story can be seen as a problem (Fido is scratching) and a solution (flea treatment). The need to make Fido happy again could be the motivation to take him to the vet. Applied recursively, these relations explain the coherence of a narrative in a hierarchical form, producing a tree. Hobbs (1985) states: "In a well-planned text, it is possible that one tree will span the entire text." This idea is propagated further and more completely in Rhetorical Structure Theory (RST) (Mann and Thompson, 1988).

RST is simple, has precise, detailed definitions for its relationships (useful for technical authors wanting some certainty that the correct relationship is being applied) and it can be used as a guide to producing coherent narratives. RST has the added benefit that it, to some extent, allows an author to gauge the quality of the narrative by considering the number of segments in it that are involved in RST relationships. Mann and Thompson also conjecture that if all the relationships can be assembled to form a tree, the narrative is likely to be coherent. This provides a way to evaluate narratives.

RST has also become one of the most popular discourse theories with a multitude of researchers experimenting with, studying, applying and implementing it. For all these reasons, RST is the theory chosen for our research. The diagram below shows one possible RST analysis of our sample story. The way to do this analysis is explained in detail in the next chapter.

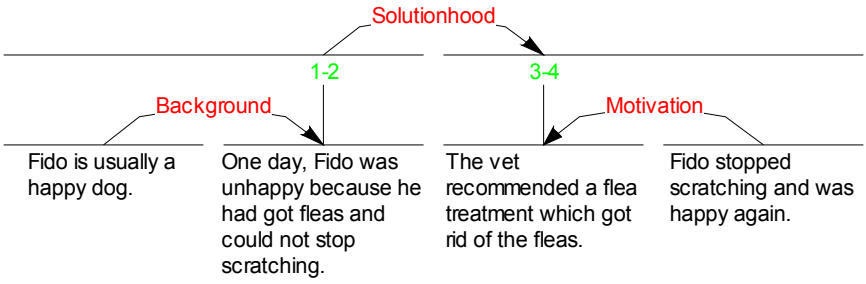


Figure 2-17: RST analysis of the Fido and Flea story

2.7 Summary

The aims of this chapter were to further the understanding of the problem we address in the thesis and provide some background material for the solution we propose in the forthcoming chapters. We allow the document narrative (DN) below to sum up the story that we had hoped to convey in this chapter.

Coherence is the attribute of a document (that is assumed free of spelling, grammatical or factual errors) which makes it easy to read and understand. It is not always easy to achieve coherence, particularly when technical authors have to work together to produce large documents. We anticipate that the use of a planning technique at the start of the writing process and an appropriate software tool can help this situation. However, the current techniques and tools available to authors do not adequately support document coherence. We then began looking at ways to fill this gap. We conjecture that coherence can be linked to the story (or narrative) that a document conveys to the reader and that enhancing this story will improve the coherence. Studies into narratives revealed that there are theories to formalise the structure of a text and make sure that it is consistent. Such a theory can help technical authors formulate better stories for their documents. After examining several possible theories, we have chosen Rhetorical Structure Theory (RST) for our research.

Figure 2-18: DN for this chapter

Chapter 3

Rhetorical Structure Theory (RST)

A discourse theory can help understand and enhance the coherence of a narrative. As seen in the previous chapter, there are several such theories. Rhetorical Structure Theory (RST), described by Mann and Thompson (1988) is one of them.

RST was created in the 1980's by a group of researchers interested in Natural Language Generation: William Mann, Christian Matthiessen and Sandra Thompson. Since it was first described, the theory has enjoyed widespread success and been used in a variety of applications ranging from teaching students to write (Mahmud and Ramsay, 2005, Mahmud, 2004) to generating puppet presentations (Rizzo et al., 2002) and Japanese abstracts (Ono et al., 1994). Other applications of RST are described in (Taboada and Mann, 2006a).

RST provides a bird's eye view of a text (Taboada and Mann, 2006b). It asserts a hierarchical structure as discussed at the end of the last chapter. Each part of the text is expected to have a purpose and be related to the other parts. A text is said to be coherent by virtue of these relationships (Reiter and Dale, 2000), particularly if the relationships can be assembled into a tree like that shown in Figure 3-1. Mann and Thompson have stated that a majority of texts appear to have a RST analysis with some known exceptions such as laws, contracts and poetry.

For several reasons, RST was a clear choice for our research. It proposes a simpler and more complete view of text organisation than most other theories. It has precise relationship definitions and the ability to help the author evaluate the level of coherence in a text. The latter is possible because RST requires a tree of relationships to be formed. If this tree cannot be produced easily, it is conjectured by Mann and Thompson that the text may not be coherent. Finally, RST is also applicable to technical documents making it ideal for our work. Therefore, this chapter describes how to analyse a text using RST.

3.1 Applying RST: First example

The first example is a paragraph from an editorial in The Hartford Courant which has already been analysed by Mann and Thompson. The text (divided into segments) and the RST analysis of it are given below.

> [Farmington police had to help control traffic recently][1] [when hundreds of people lined up to be among the first applying for jobs at the yet-to-open Marriott Hotel.][2] [The hotel's help-wanted announcement - for 300 openings - was a rare opportunity for many unemployed.][3] [The people waiting in line carried a message, a refutation, of claims that the jobless could be employed if only they showed enough moxie.][4] [Every rule has exceptions,][5] [but the tragic and too-common tableaux of hundreds or even thousands of people snake-lining up for any task with a paycheck illustrates a lack of jobs,][6] [not laziness.][7]

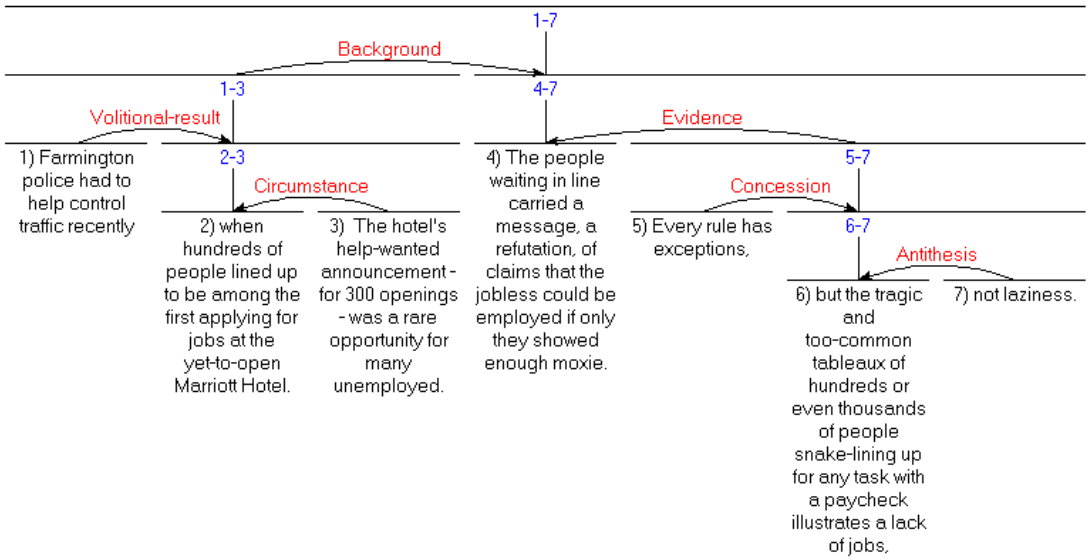


Figure 3-1: First paragraph of an editorial in The Hartford Courant (above) and the RST analysis for it (below) done by Mann and Thompson

A successful RST analysis is expected to result in a Rhetorical Structure tree (RS-tree) as shown above. Many of the RS-trees in this thesis have been drawn using the free software tool called RSTTool (O'Donnell, 2000). The way in which they are drawn may seem unconventional to computer scientists but this was the way that Mann and Thompson chose to draw them and we have adhered to their style.

These RS-trees are similar to traditional tree structures except that they have additional RST relationships added on. So, in the diagram above, the root of the tree is the node containing segments 1 to 7 from the text. This is divided into two subtrees: 1-3 and 4-7. The tree structure is actually denoted by the horizontal lines. Segments 1-3 are joined by a

BACKGROUND relationship to segments 4-7. In this case, segments 1-3 form what is called the "satellite" of the BACKGROUND relationship and segments 4-7 form the "nucleus". These relationships have precise definitions (see Table 3-1 for one such definition).

The subtree containing segments 4-7 is further divided into two: 4 and 5-7. Segments 5-7 (satellite) are linked by an EVIDENCE relationship to segment 4 (nucleus). This continues until all the segments are connected by relationships. A more traditional tree-view of this RS-tree is given in section 3.3.3. The process of doing this analysis is explained in detail in the forthcoming sections.

Relation name:	**EVIDENCE**
Constraints on the Nucleus (N):	The reader R might not believe the information that is conveyed by the nucleus N to a degree satisfactory to the writer W
Constraints on the Satellite (S):	The reader believes the information that is conveyed by the satellite S or will find it credible
Constraints on N+S combination:	R's comprehending S increases the R's belief of N
The effect:	R's belief of N is increased
Locus of the effect:	N

Table 3-1: The definition of the EVIDENCE relationship in RST (Mann and Thompson). Note that segment 4 in the analysis above is the nucleus of the EVIDENCE relationship and segments 5-7 together form the satellite. The other relationship definitions are in Appendix A.

3.2 First step: Segmentation

The first step in the analysis is dividing the text into non-overlapping **segments**[5]. In Figure 3-1 the segments were demarcated using square brackets. The segmentation is done prior to the analysis to avoid circularities (i.e. an analysis depending on the segments and segment choices depending on the analysis) (Taboada and Mann, 2006b). Each segment is required to have independent functional integrity. Therefore, the segments in Mann and Thompson's analyses are often clauses[6].

RST establishes two types of segments in a text: **nuclei** and **satellites**. Nuclei are segments that are most important and essential to the understanding of the text. Satellites contribute to the understanding of the nuclei but are secondary. Therefore, a text without its satellites should still be comprehensible (like a synopsis of the original text) but not a text without its nuclei. Going back to the EVIDENCE relationship in Figure 3-1, segment 4 is the nucleus. If read on its own, it still manages to convey most of its message: that unemployment is, perhaps, not entirely due to the laziness of the people. Segments 5-7 provide additional

[5] In some other descriptions of RST, segments are also called units.
[6] A clause is a group of words containing a subject and a predicate, usually a part of a more complex sentence.

evidence and information for this statement. However, on their own, the purpose of segments 5-7 would not be entirely clear.

3.2.1 Segment size

The size of a segment is arbitrary. Very large segments have been discouraged by Mann and Thompson since there could be units within a large segment that belong to relationships with units outside that segment. However, large segments are not entirely uncommon, particularly when studying the overall structure of bigger texts (Taboada and Mann, 2006b, Mann et al., 1992).

In our research, RST is applied to relatively short texts called document narratives (DN). So, the segments will often be clauses or, at most, a few sentences. These segments in the DN will eventually correspond to sections or chapters of a large document. Chapter 4 explains, in detail, this new technique that we propose.

3.3 Second step: Defining the RST relationships

The second step is to define relationships between the segments. A relationship identifies a clearly established connection between two (or more) segments. In RST, only *one* relationship can be applied to a pair of segments. Even though there have been arguments against this restriction (Moore and Pollack, 1992), we will obey Mann and Thompson's rule.

Most relationships are between a nucleus (N) and a satellite (S). Mann and Thompson calls these relationships **hypotactic**. A few relationships, such as SEQUENCE and CONTRAST, exist between segments of equal importance. They are called multi-nuclear or **paratactic** relationships (Mann and Thompson). Not all relationships are binary either. The SEQUENCE relationship can be applied to as many segments as necessary.

There is also a JOINT schema (see section 3.3.1 for other schemas) which can be applied to multiple segments, but it is unclear when one would use JOINT. Mann and Thompson state that it is the declared *absence* of a relationship and, in one of their analyses, have used it between the segments listed below. They are adjacent segments from a letter persuading readers to donate money to an organisation (Mann et al., 1992).

1. Our small staff is being swamped with requests for more information
2. and our modest resources are being stretched to the limit.

We have, so far, not used JOINT in the analyses we have done. Mann and Thompson draw hypotactic and paratactic relationships as illustrated below. In these diagrams, nuclei are represented under vertical (or diagonal) lines and the relationships are denoted by labelled curved lines. In figures showing hypotactic relationships, the curved lines are arrows that always point towards the nucleus.

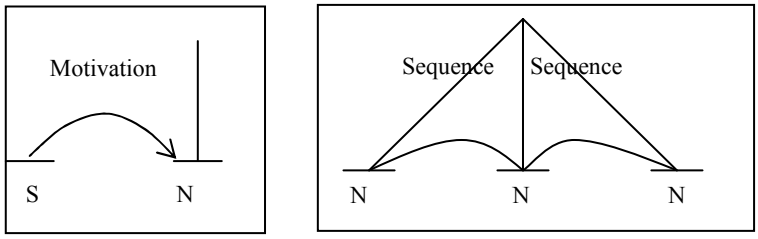


Figure 3-2: An illustration of a hypotactic relationship (left) and a paratactic relationship (right). The curved lines are labelled with the name of the relationship. In a hypotactic relationship, the arrowhead always points to the nucleus. Nuclei are indicated by vertical (or diagonal) lines above them.

Mann and Thompson identified 23[♦] relationships in their original paper. They are listed below. Mann and Thompson emphasised, however, that this is in no way a closed list and expected additions for different genres of text. In our analysis of technical documents, only a few of these relationships have been used regularly. These are marked with an asterisk. This will be discussed in the next chapter.

Hypotactic relations	12. Purpose
	13. Antithesis
1. Evidence*	14. Concession
2. Justify*	15. Condition
3. Solutionhood*	16. Otherwise
4. Elaboration*	17. Interpretation
5. Background*	18. Evaluation
6. Enablement*	19. Restatement
7. Motivation*	20. Summary
8. Circumstance	
9. Volitional Cause	***Paratactic relations***
10. Non-Volitional Cause	
11. Volitional Result	21. Sequence*
Non-Volitional Result	22. Contrast*

Figure 3-3: List of all 23 relationships. The ones used regularly in our research have been marked with an asterisk

Each relationship has a precise definition for its nucleus, satellite, their combination and the effect it has on the reader. The definition for the EVIDENCE relationship was given in Table 3-1. The definitions for the other relationships are in Appendix A (section A.1).

3.3.1 Five schemas

There are five structures (or **schemas**) according to which the relationships can be applied. Schemas specify how text segments can co-occur. For instance, a CONTRAST schema should always have exactly two nuclei. The other schemas are represented by the examples below. JOINT, a multinuclear schema, has no corresponding relationship.

[♦] 23 relationships and JOINT (a multi-nuclear schema) which is the declared absence of a relationship

Schemas for relationships not shown in the figures all follow the simple pattern represented by the CIRCUMSTANCE relationship: a single relationship with a nucleus and a satellite (Mann and Thompson).

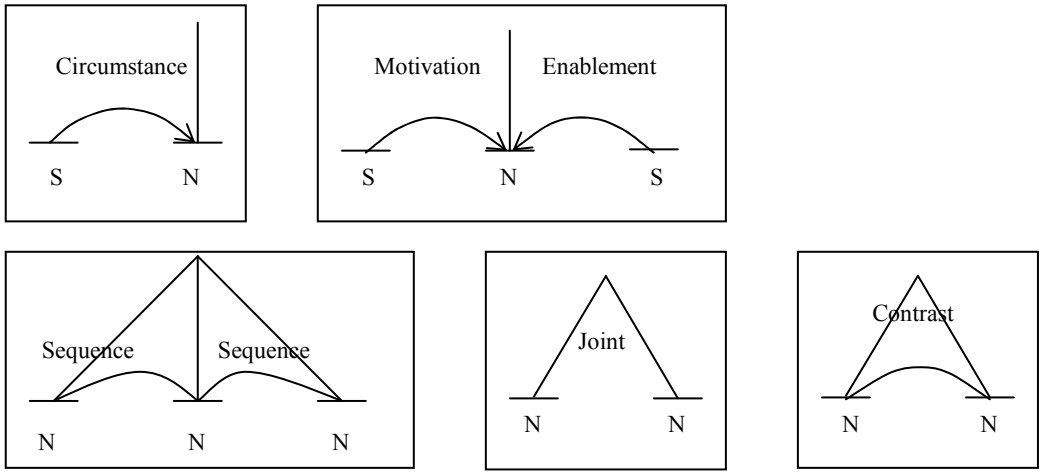


Figure 3-4: Five types of schemas in RST

3.3.2 Recognising the relationships

Some relationships in a text are *signalled* explicitly by cue phrases such as conjunctions, and the mood and tense of the text. For instance, two relationships in the analysis in Figure 3-1 are signalled by the cue words below.

- *when* (in segment 2, helping to signal a CIRCUMSTANCE)
- *but* (in segment 6, helping to signal an ANTITHESIS)

A corpus of other cue words and phrases that signal relationships can be found in Appendix A of Alistair Knott's PhD thesis (Knott, 1996).

However, many relationships can also be identified without the presence of explicit signals (Taboada and Mann, 2006b). For example, the remaining relationships in the analysis in Figure 3-1 - VOLITIONAL-RESULT, BACKGROUND, EVIDENCE and CONCESSION – had been considered to exist in the text by Mann and Thompson without any obvious signals.

In our use of RST, the analyst is often also the author of the text. Therefore, having created the text with a certain understanding of it, it is anticipated that the analyst cum author would not need to heavily rely on signals to recognise the relationships.

3.3.3 Forming the RS-tree

Relationships need to be defined recursively working either from the top down or from the bottom up, or both, as deemed convenient. The segments joined by a relationship form a **span**[7] which can, in turn, become part of another relationship (hence, recursive). As an example, consider the relationships in the analysis in Figure 3-1. Segments 2 and 3 are joined using a CIRCUMSTANCE relationship. The span 2-3 then goes on to become part of the VOLITIONAL-RESULT relationship with segment 1.

This continues until the relationship schemas can be assembled into a **rhetorical structure tree** (RS-tree). The *tree* in RS-tree structures like the one in Figure 3-1 may not always be easily recognisable. Therefore, the diagram below shows a different view of the RS-tree in Figure 3-1. The traditional tree structure is in black and the RST relationships are in blue. The names of the relationships have not been included so as not to clutter the diagram.

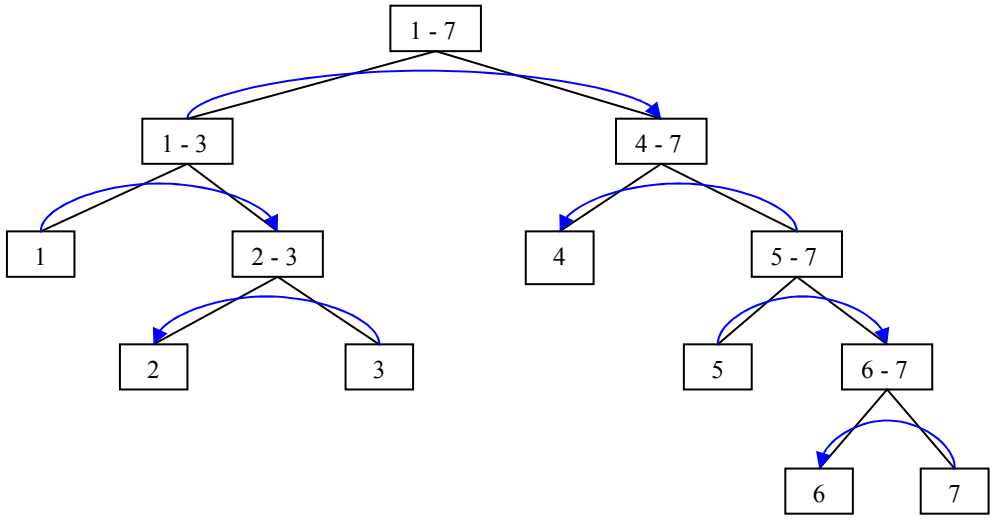


Figure 3-5: RS-tree from Figure 3-1 redrawn to highlight the tree structure

There can be, as one might imagine, more than one RS-tree for a given text. This is why, whenever we present a RST analysis of a text in this thesis, we label it as "one possible analysis". Analysts may identify different relationships between the same pair of segments or group segments in different ways. However, the important point to be noted is that, no matter what the individual relationships are, the eventual RS-tree has to be *well-formed* (see below). A text that forms a well-formed RS-tree when analysed is expected to be coherent.

[7] We distinguish between a segment and a span. Most RST descriptions just use 'text spans' to refer to all participants in a relationship.

There are four criteria that determine whether a RS-tree is well-formed (Mann and Thompson, 1988, Marcu, 2000). They are listed and explained below. Each criterion is examined with respect to the well-formed tree in Figure 3-1 (and Figure 3-5).

Completedness: One schema application (the root) should cover the entire text. For example, in Figure 3-1, the BACKGROUND relationship at the top includes all the seven segments of the 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. In other words, each node in the tree, apart from the root, must either be a leaf node or an internal node. Segments that cannot fit in the tree structure via relationships are referred to as non-sequiturs and are a sign of a lack of coherence in the text.

Uniqueness: Each text span/segment should have only one parent (i.e. each schema application consists of a different set of text spans/segments). So, for instance, in Figure 3-5 above, the "parent" of segment 4 is the span 4-7. It cannot also be a child of another span as illustrated below.

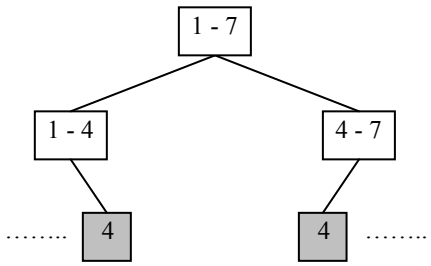


Figure 3-6: Invalid application of RST. Segment 4 has more than one parent.

Adjacency: Only adjacent text spans/segments can be grouped together to form larger spans. So, in the example in Figure 3-5, segment 5 and span 6-7 have been grouped to form a larger span 5-7 because they are adjacent. Span 6-7 cannot be joined with segment 4, say, to form a span. It appears that individual relationships, however, can exist between non-adjacent segments or spans, provided that they have the same parent. Looking ahead to Figure 3-15, for instance, segment 9 is linked to segment 5 which is not adjacent to it.

3.4 RST and text coherence

One of the main reasons we chose RST for our research was because of its ability to help authors *evaluate* the level of coherence in a text. Coherence in RST arises due to the set of constraints associated with each relationship and the overall effect on the reader. For instance, in a MOTIVATION relationship, the reader expects to find in the satellite some information that will persuade him to perform the action presented in the nucleus. These expectations are dictated by the relationship definitions (see Appendix A for all the definitions). The more segments that can be linked via relationships, the better the quality of the narrative.

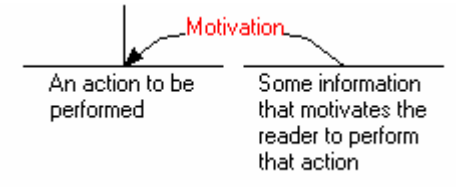


Figure 3-7: A MOTIVATION relationship

It is also strongly conjectured that the construction of a well-formed RS-tree for a given text would suggest that it is coherent. "If an RST diagram is a connected whole, with every unit of the text linked into the diagram somehow, then the analysis demonstrates how the text can be seen as coherent" (Taboada and Mann, 2006b). It is almost a test for coherence. The presence of non-sequiturs (segments of text that do not seem to belong) suggests a lack of coherence. This too helps authors gauge the quality of their narratives.

To demonstrate this feature, the example of coherent and incoherent text from Chapter 2 has been considered again. The texts have been reproduced below for convenience.

The World in 1993

1993 will start with the world in a pessimistic frame of mind. That gloom should soon dispel itself. A clear economic recovery is under way. Though it will be hesitant at first, it will last the longer for being so. If you are sitting in one of the world's blackspots, this prediction will seem hopelessly optimistic. But next year's wealth won't return to yesteryear's winners; these middle-aged rich people need to look over their shoulders to the younger world that is closing in on them.

The World in 1993

1993 will start with the world in a pessimistic frame of mind. A clear economic recovery is under way. That gloom should soon dispel itself. These middle-aged rich people need to look over their shoulders to the younger world that is closing in on them. But next year's wealth won't return to yesteryear's winners; it will last the longer for being so if you are sitting in one of the world's blackspots. Though it will be hesitant at first, this prediction will seem hopelessly optimistic.

Figure 3-8: An example of a coherent (left) and an incoherent (right) text. Source: (Knott, 1996)

It was possible to create a RS-tree for the coherent text.

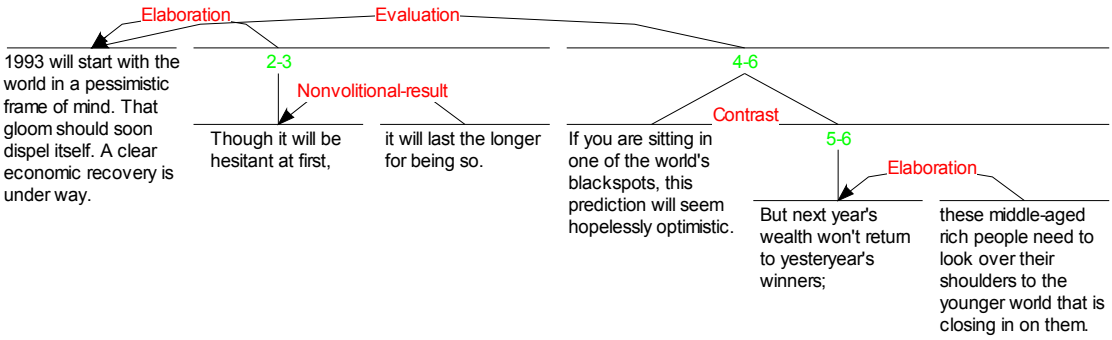


Figure 3-9: A possible RS-tree for the coherent text from Figure 3-8

It was, however, not possible to complete an analysis for the incoherent text (Figure 3-10); thus showing that RST's requirement of a tree structure is a valuable tool to evaluate the level of coherence in a text.

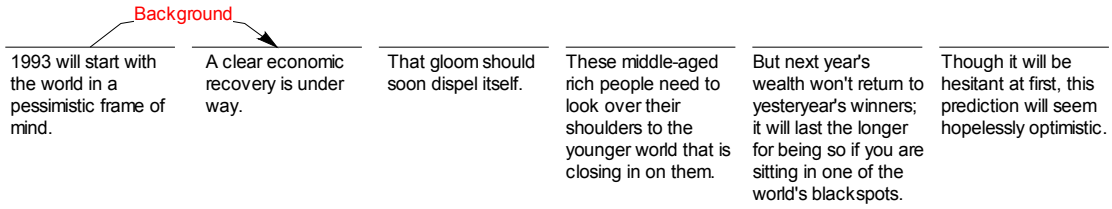


Figure 3-10: An attempt to analyse the incoherent text from Figure 3-8. There were no clear relationships between the segments.

RST also provides some guidance to the author about how to structure the text. For instance, Mann and Thompson recommend the order of the satellite and nucleus for some relationships. The order of text spans is not usually constrained by the relationship and is under the control of the author. However, after analysing many texts, Mann and Thompson identified patterns for some relationships. It has been observed that this ordering often improves the quality of the text. For instance, placing the problem (satellite) before the solution (nucleus) in a SOLUTIONHOOD relationship is generally considered better and more coherent. Similarly, for a BACKGROUND relationship, it was suggested that the background material (satellite) be presented before the nucleus. Both these examples are common practice in technical documents. There were other recommendations too (listed below).

Satellite before Nucleus		
	Antithesis	Conditional
	Background	Justify
	Concession	Solutionhood
Nucleus before Satellite		
	Elaboration	Purpose
	Enablement	Restatement
	Evidence	

Figure 3-11: The orders identified by Mann and Thompson for some relationships

3.5 Applying RST: Second example

RST is applied to a second example to reinforce the information in the preceding sections. The text analysed is an article from the BBC website[8]. The article has been shortened to provide a suitably sized example. The text was divided into nine segments as shown below.

China introduces chopsticks tax

[China produces about 45 billion pairs of chopsticks a year, consuming millions of trees and bamboo plants.][1] [The disposable splints of wood, usually between eight and 10 inches long, have long been a target for Chinese environmentalists.][2]

[School children have written to the Chinese prime minister asking for a ban on disposable wooden chopsticks, while students have persuaded some college cafeterias to replace them with spoons.][3] [In recent years, the government has actually encouraged their use, in a bid to reduce the spread of infectious illnesses by sharing eating utensils.][4]

[The Chinese government is introducing a 5% tax on disposable wooden chopsticks in a bid to preserve its forests.][5] [The move came as China said it would raise some consumption taxes next month in a bid to help the environment and narrow the gap between rich and poor.][6]

["This is part of the government's strategy of rebalancing growth and reducing energy demand," said Ben Simpfendorfer, a strategist with the Royal Bank of Scotland in Hong Kong.][7] ["The government wants to show that it is doing something to increase the tax burden on the richer segment of the population to reduce the widening disparity between the rich and poor."][8]

[Shanghai consumers gave a mixed response to the new tax.][9]

Figure 3-12: An article from the BBC website (shortened) divided into segments

The RST analysis was done bottom up. So, segment 2 was seen as providing background information to the problem stated in segment 1. Segments 3 and 4 were recognised as being in a CONTRAST relationship. Similarly, segments 7 and 8 were linked by a SEQUENCE relationship and the span 7-8 then became part of an ELABORATION relationship with 6. The remarks by the strategist at the end of the article were seen as elaborations of the plan by the Chinese government to help the environment and community. These subtrees are shown below.

[8] The original article was published on the website on the 22nd of March, 2006. It can be found at: http://news.bbc.co.uk/1/hi/business/4831734.stm (Last accessed on the 17th of August, 2006).

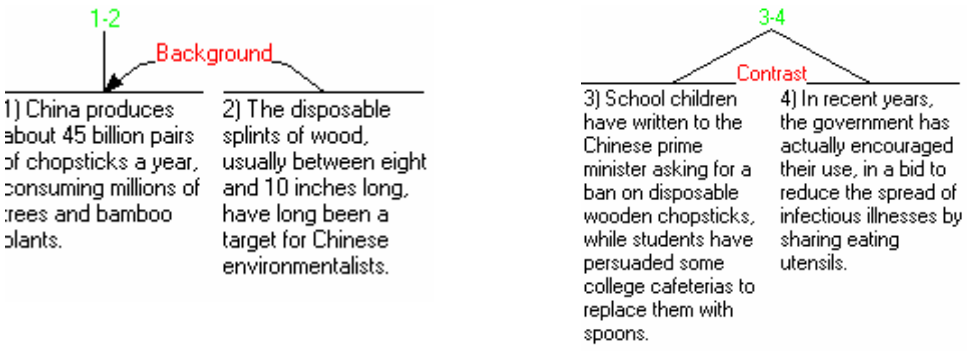


Figure 3-13: Segments 1 and 2 in a BACKGROUND relationship, and segments 3 and 4 in a CONTRAST relationship.

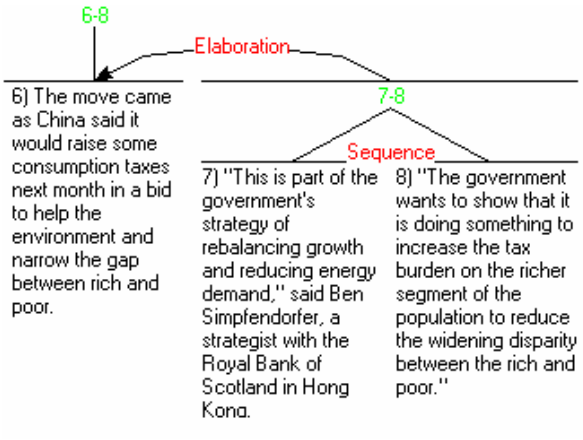


Figure 3-14: An ELABORATION and SEQUENCE relationship

The whole RS-tree thus formed is illustrated below. The subtrees involving segments 1-2, 3-4 and 6-8 that have already been shown are collapsed in order to make the other segments more visible. The whole article is seen as the presentation of a problem and a possible solution to it. Hence, a SOLUTIONHOOD relationship is used to cover the entire text.

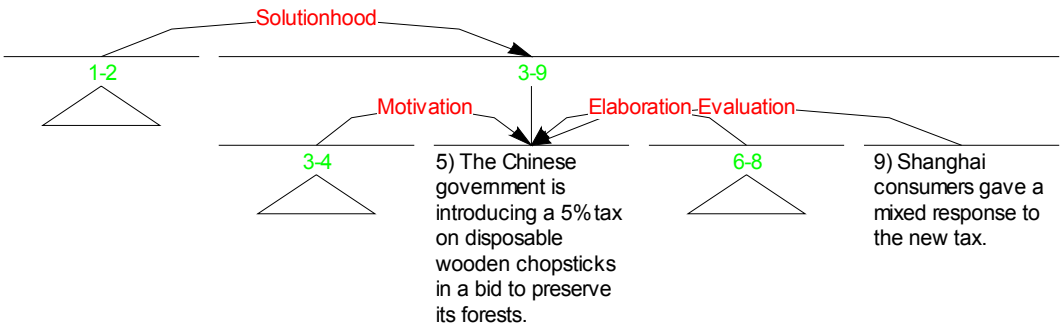


Figure 3-15: The complete RS-tree for the BBC news article. Note that segment 9 is in a relationship with non-adjacent segment 5. This is allowed in RST. It is the joining of non-adjacent segments or spans to form larger spans that is not allowed in well-formed RS-trees.

3.6 Summary

RST is a discourse theory that asserts a hierarchical structure in a text. Parts of the text are linked to other parts with relationships such as MOTIVATION and SEQUENCE. A text is said to be coherent if the relationships within it can be assembled into a well-formed RS-tree.

Mann and Thompson and other researchers have analysed large corpuses of texts. One such corpus includes the analysis of 385 Wall Street Journals. The corpus was produced by Carlson et al. (Carlson et al., 2001) and is now distributed by the Linguistic Data Consortium (LDC). Studying some of these analyses has helped us understand RST better[9].

Some researchers into RST have commented on various shortcomings of the theory. For instance, Moore and Pollack (1992) strongly contradict the rule of having just one relationship between two segments. They believe that there needs to be multiple, co-existing levels of analysis (i.e. more than one relationship between segments) if a model of discourse is to be complete. One might ask if one level of analysis is adequate for our research. The answer, for now, is yes. Even though more relationships would perhaps indicate a higher level of coherence, multiple levels of analysis would delay technical authors and complicate the process that we propose in Chapter 4.

RST exhibits many of the properties that we were looking for in a discourse theory. This chapter is a tutorial on RST and marks the end of the background material in this thesis. From this point forward, we introduce the original contributions of our research. Chapter 4 presents the new technique we propose called narrative-based writing that makes use of RST. Chapters 5 and 7 describe the design and implementation of a tool that helps co-authors improve the coherence of technical documents by using narrative-based writing. The remainder of the chapters evaluate and show applications of this technique and tool.

The previous chapter discussed a number of discourse theories. RST is one of them. Since its creation in the 1980's, RST has enjoyed widespread use in a variety of applications beyond what it was originally built for. We too find that RST is the most suitable for our research for a number of reasons. The process of applying RST involves two steps: segmentation and the recursive definition of relationships between spans, constructing a well-formed RS-tree. There are five schemas and four criteria that determine whether a RS-tree is well-formed. It is popularly conjectured that only a coherent text can produce a well-formed RS-tree. Over the years, researchers have pointed out shortcomings in the theory but we find RST to be adequate for our research into the coherence of technical documents.

Figure 3-16: DN for this chapter

[9] More information about RST can be found in M&T's original paper or the website about RST that is maintained by Maite Taboada (http://www.sfu.ca/rst/).

Chapter 4

Narrative-based writing:
A new approach to document planning

To sum up so far: Coherence is the feature of a document that makes it easy to read and understand. Technical documents, unfortunately, have a reputation for being poorly structured and incoherent. One of the main reasons for this is that these documents are often produced collaboratively with several authors contributing sections which are collated to produce the final version. We found that current collaborative writings tools and planning techniques available to technical authors, while being good at what they were designed for, did not provide sufficient support for coherence. Our goals, therefore, are to:

1. Devise a planning technique that improves document coherence
2. Develop a tool that helps teams of co-authors use this technique

This chapter addresses the first of these goals. Chapters 5 and 6 address the second.

Halfway through the background chapter (chapter 2), we introduced the idea of linking document coherence to the implicit story conveyed to the reader. Based on this, we propose a new technique called **narrative-based writing.** It brings together these previously parallel strands of research: technical documents, coherence, narratives and RST.

In the new technique, a précis of the story conveyed by a document is called a **document narrative (DN).** Thinking about the suitable DN for a document helps an author arrange the content of a document in an appropriate fashion. As additional verification of the quality of the DN, RST can be applied to it. The RST relationships also make clear the authors' intentions for creating the narrative in a certain way (e.g. Motivation, Justify). Finally, the DN and the RST analysis can be used to write the document.

The steps in narrative-based writing are explained and illustrated in this chapter.

4.1 The technique explained

Narrative-based writing is the new technique we propose to assist authors in planning coherent documents. The technique stems from the idea that the coherence of a document can be attributed to the story conveyed to the reader, which we call the DN. The process can be distilled into these three steps:

1. Formulate the DN
2. Analyse the DN using RST to study and gauge its coherence
3. Use the DN and RST analysis as a guide to structuring the document

Each of the steps are described below. We use chapter 3 of this thesis as an example to demonstrate each step.

4.1.1 Formulating the document narrative (DN)

The first step in this planning technique is to write down the DN. A DN is an explicit précis of the story that a document conveys to the reader. We sometimes use the analogy of an *elevator speech* or an *executive summary* to describe what a DN is. It is a top-level view of what the document is expected to say to the reader.

With most writing tasks, authors often start out with a list of things they want to include in the document. So, for instance, the ideas that were considered important (in no particular order) for chapter 3 were:

History and overview of RST
Shortcomings of RST
Why was it chosen over the other theories?
How to do the RST analysis – describe each step
One or two examples

An author will generally ponder on the ideas for a while before "putting pen to paper". However, in what order should the ideas be presented and how should they be linked together? What is the story that the document should convey? For experienced authors this is often a straightforward task. Others may, time permitting, try various combinations and revise the document until it 'sounds right'. For others still, this is not trivial.

Narrative-based writing provides a way to get this 'story straight' right from the beginning. By having to think explicitly about the DN, authors iron out inconsistencies and link bits of the content together in a natural way. If a piece of information is really difficult to fit into the DN, it may be an indication for it to be left out of the document or re-inserted in a different location. A DN gives technical authors a quick way to formulate the story for their document. This is beneficial for technical authors who often have a short time to plan this story as opposed to say, novelists, who may take years to plan a novel.

Formulating the DN from a list of ideas may not, at once, be obvious. A certain amount of persistence is necessary to arrive at the best DN, having ruminated on everything that needs to be said in the document. The DN used for Chapter 3, generated from the list of key ideas above, is shown in Figure 4-1.

[The previous chapter discussed a number of discourse theories. RST is one of them. Since its creation in the 1980's, RST has enjoyed widespread use in a variety of applications beyond what it was originally built for. We too find that RST is the most suitable for our research for a number of reasons.][1] [The process of applying RST involves two steps:][2] [segmentation][3] [and the recursive definition of relationships between spans, constructing a well-formed RS-tree.][4] [There are five schemas and four criteria that determine whether a RS-tree is well-formed.][5] [It is popularly conjectured that only a coherent text can produce a well-formed RS-tree.][6] [Over the years, researchers have pointed out shortcomings in the theory][7] [but we find RST to be adequate for our research into the coherence of technical documents.][8]

Figure 4-1: DN for Chapter 3 of this thesis

The concept of the DN has been continually refined. In our early attempts at writing them, we included the authors' intentions and reasoning as *part* of the DN. For instance, 'we want to motivate the reader to fund us' was the starting sentence in our first DN for a research proposal (De-Silva and Henderson, 2005). We used to also include structural information such as 'on the next page' or 'in one or two sentences.' An example of this previous kind of DN is given in section 4.3. However, this information made the DN difficult to read. We also realised that the RST relationships captured the author's intentions, making it unnecessary to repeat them in the DN. Both these practices have now been abandoned and DNs focus purely on the story that gets across to the reader.

We recommend that the DNs be kept relatively short. Each of our DNs is usually no more than half a page long. This is because a DN is meant to be a top-level view of a document and is not expected to contain much detail. It is also expected to provide the author with a mental model of the document which he can think about and improve. It would be difficult to do this with a large DN[10].

Our recommendation is to create a high level DN for the document and then to create DNs for each of the chapters if more detail is required. This is similar to the concept of a 'framing narrative' (Abbott, 2002). A framing narrative is one that contains (and puts into context) other narratives. Abbott uses the tale of the 'One thousand and One Arabian nights' (McCaughrean, 1999) as an example of a framing narrative. In order to delay her execution, Queen Shaharazad tells her murdering husband a wonderfully exciting story every night. The

[10] It has been shown that humans can only hold a certain number of concepts in the mind at one time (e.g. 'seven plus or minus two' theory).

king is used to a new wife every day only to put her to death the next morning. However, he becomes so intrigued with Shaharazad's stories that he keeps postponing her execution. Each of the stories told by the queen is contained within this framing narrative. In some ways, say in a book or thesis for example, the DN for the whole document is similar to a framing narrative and the DN for each of the chapters is a story in itself but contained within the main DN.

Having got a DN as a guide, authors can proceed to write the document. However, we conjecture that the more coherent the DN is, the more coherent the document will be. Therefore, as an additional step, we use RST to study, improve and validate the coherence of the DN before using it to plan the document.

4.1.2 Analysing the DN using RST

The second step in narrative-based writing is the RST analysis. There are several properties of RST that will help improve the structure and coherence of a DN as explained in the previous chapter. If a well-formed tree can be constructed for a DN, the authors have some assurance of the quality of their DN.

The DN for Chapter 3 was segmented as shown (demarcated by the square brackets) in Figure 4-1. A RST analysis for it is shown in the figures below. It was not possible to fit the entire diagram onto the page, so the RS-tree has been presented in parts. One of the first relationships that can be identified is the SEQUENCE relationship between segments 3 and 4. However, segments 5 and 6 elaborate the process of creating the RS-tree. Hence, the subtree below is a possible analysis of segments 3-6.

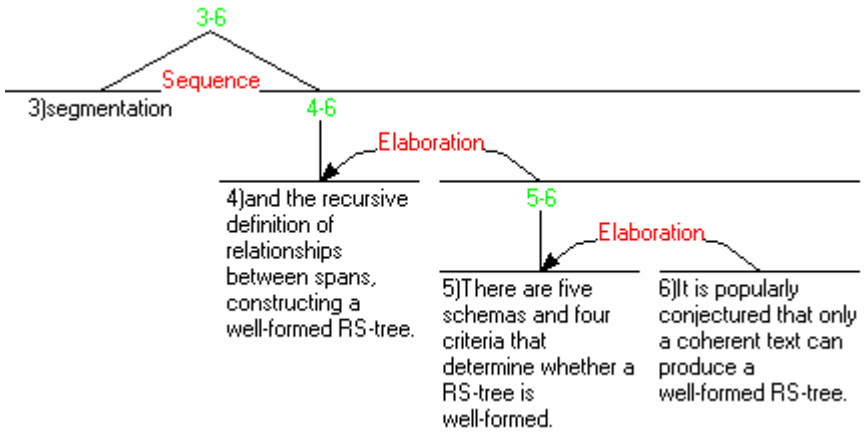


Figure 4-2: Sub tree showing segments 3-6

Secondly, it was observed that segments 7 and 8 are in a contrast relationship.

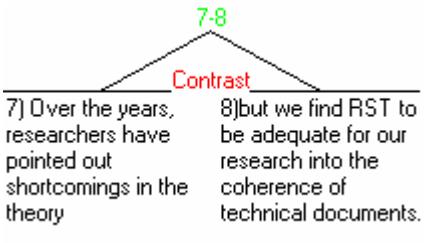


Figure 4-3: Subtree showing segments 7 and 8

These subtrees are linked to the rest of the segments as shown below. Once again, it needs to be stressed that this is just *one* of the possible analyses of this DN. The important point is that a well-formed RS-tree was constructed from the segments. Note that the span 7-8 is linked to segment 2 by an ELABORATION relationship even though they are not adjacent to each other. This is, as mentioned in Chapter 3, allowed in well-formed RS-trees. However, non-adjacent spans cannot be joined together to form larger spans. So, segment 2 and the span 7-8 cannot be grouped together to become part of another relationship.

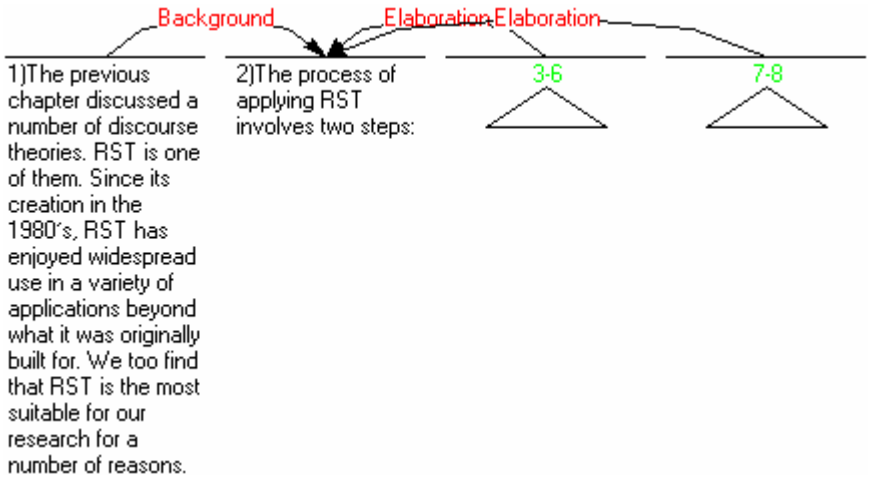


Figure 4-4: Possible RST analysis of the DN for chapter 3

Mann and Thompson identified 23 RST relationships. It is a tall order to expect a technical author to remember the definitions for each of these relationships. Therefore, we have identified a subset of nine relationships that have been consistently used in the analyses done so far on technical documents. We list these relationships below along with a summarised definition for each. We do not claim that this subset is sufficient for *all* technical documents. More analyses need to be done (by different people) to establish if this is the case. In addition to this nine, two other relationships have been used too but only occasionally. They are all listed below.

The following tables list the set of relationships used frequently in our analyses along with a brief description. The relationships for which Mann and Thompson recommended the order of the satellite (S) and nucleus (N) are also indicated. The relationships for which there were no explicit recommendations have been left blank.

Name	Description	Order of N & S
Background	Satellite provides background information to the nucleus	S before N
Contrast	Applies to two nuclei that contrast each other	
Elaboration	Satellite elaborates the information in the nucleus	N before S
Enablement	Information in the satellite enables the reader to perform action in nucleus	N before S
Evidence	Satellite provides evidence to the statement in the nucleus	N before S
Justify	Satellite justifies the nucleus	
Motivation	Satellite motivates the reader to perform the action in the nucleus	
Sequence	Multiple nuclei that follow each other in sequence	
Solutionhood	Satellite is the problem. Nucleus provides the solution.	S before N

Table 4-1: Subset of relationships used frequently to analyse technical documents

Name	Description	Order of N & S
Purpose	The nucleus presents an intended situation and the satellite presents the intent behind that situation	N before S
Volitional-result	The satellite is a result of the action in the nucleus	

Table 4-2: Two relationships used less frequently

A RST relationship also encapsulates the authors' intentions and reasoning for having certain segments of the DN. In a collaborative writing scenario, this is an ideal way to communicate these intentions for the document. Since there are detailed definitions for each of the relationships, there is little room for ambiguity about what a specific relationship means. (The role narrative-based writing can play in collaborative writing is discussed in section 4.2.) RST also provides a mechanism for rationalising a DN. Authors in a team may have different opinions of what the DN should be for their document and using RST can help them justify their choices and come to an agreement. Having analysed the DN, this information can now be used to structure and produce the document. This is explained next.

4.1.3 Producing the document

The final step in narrative-based writing is using the DN and the RST analysis as a guide to structuring the document. We recognise two ways in which the document can directly be influenced:

1. The *sequence* of ideas as they appear in the DN is a good indication of the sequence that they should be presented in the document.
2. The RST relationships can dictate how the text and examples of a particular section are crafted in order to create the anticipated effect on the reader.

Both these points are described below.

(1) We examine the positioning of sections first. When constructing the document, presenting the information in the right, logical order is vital. With narrative-based writing, the sequence of the sections in the document should correspond to the sequence of segments in the DN.

Reverting to the previous example, the sections in Chapter 3 of this thesis were organised as shown below. The corresponding segments in the DN are shown too (on the left). Note that some DN segments do not have associated sections. This is normal. These segments are just needed to glue the story together and it is likely that the other sections in the document will have information relating to these segments.

1	The previous chapter discussed a number of discourse theories. RST is one of them. Since its creation in the 1980's, RST has enjoyed widespread use in a variety of applications beyond what it was originally built for. We too find that RST is the most suitable for our research for a number of reasons.	*Introduction to the chapter
2	The process of applying RST involves two steps:	
3	Segmentation	*Segmentation
4	and the recursive definition of relationships between spans, constructing a well-formed RS-tree.	*Defining relationships
5	There are five schemas and four criteria that determine whether a RS-tree is well-formed.	
6	It is popularly conjectured that only a coherent text can produce a well-formed RS-tree.	*RST and text coherence
7	Over the years, researchers have pointed out shortcomings in the theory	*Summary of the chapter
8	but we find RST to be adequate for our research into the coherence of technical documents.	

Table 4-3: Correspondence between DN segments and sections of Chapter 3

Certain sections or chapters in a document are required due to standard practice. For instance, most documents are required to have an introduction at the start and a conclusion at the end. Similarly, letters are expected to have a letterhead and a signature. These fixed structures are sometimes called holistic structures (Mann et al., 1992). Narrative-based writing is a way of planning the rest of the document; the relational aspects of the *body* of the document.

The suggested orders of the satellites and nuclei for some relationships (listed in Chapter 3) can also guide how the sections are placed. For example, it is better to present the satellite of a BACKGROUND relationship first. However, this would also require going back and changing the DN accordingly (so that the segments are in the right order).

Some ideas can also be placed in subsections depending on the sorts of relationships they are involved in. In the RST analysis above, segment 5 is an elaboration of segment 4. Therefore, we decided to place the sections about the schemas and the four criteria of RS-trees (segment 5) as subsections of the section about defining relationships (segment 4). In our experience, this is often a reasonable practice for most situations. If the nucleus of an ELABORATION relationship is a section, then the satellite can be the subsection. We have defied this rule when we included a section called 'RST and text coherence' as a section on its own, even though the corresponding segment in the DN is a satellite of an ELABORATION relationship. So, it is important to note that these suggestions are guidelines, not fixed rules.

(2) We see a second way by which the document is influenced by the RST analysis: the need to establish the appropriate RST relationship in and across the sections. So, a section corresponding to a DN segment involved in a MOTIVATION relationship, for instance, needs to motivate the reader to perform the actions in the other sections (or say why the authors were motivated to perform those actions) and so on. This is usually done by crafting the text accordingly, by choosing the right examples and so on. So, in chapter 3, we introduced two examples of an RST analysis to contribute to the BACKGROUND and ELABORATION relationships. While bringing out the relationship within the text, the section should also establish linkage and context (its connection to previous sentences/sections and so on). Once again, these are just guidelines.

3.0 (Introduction to the chapter)
3.1 Applying RST: First example
3.2 First step: Segmentation
3.3 Second step: Defining the RST relationships
3.4 RST and text coherence
3.5 Summary

Figure 4-5: Sections of Chapter 3

Eventually, the sections in Chapter 3 were presented in the order shown above. In our opinion, writing a document this way will have the most benefits in a collaborative writing scenario. This is explained in the following section.

4.2 The role of DNs in collaborative writing

The main goal of our research is to address the issue of coherence in collaborative technical writing. Narrative-based writing, in our opinion, can help alleviate this situation. The process of collaborative writing can be augmented by a DN as shown below.

1. Authors agree on and analyse a DN for the document
2. Authors formulate DNs for the individual sections if necessary
3. Assign sections and the corresponding DN to authors
4. Each author, now aware of all the narrative goals, completes the section and returns it to the team leader
5. The team leader puts the sections together so that it fits the DN for the entire document

It is anticipated that authors would first agree on and maybe analyse a DN for the whole document. This will help iron out conflicting ideas and support the structure for the document. Next, if the sections are to be assigned to various authors, DNs can be produced for the individual sections too. In a large project, these DNs can even be made by the team leader and distributed to the relevant subordinate authors.

Each author then has the responsibility of creating his section according to the DN and the RST relationships. Each section should establish links with all the corresponding ideas in the DN and bring to the surface the RST relationships. Finally, the sections can be returned to the team leader who will collate them to form the final document. In theory, the sections should fit the overall DN better.

RST relationships also help communicate ideas about the structure for the document. Since the relationships have fixed definitions, an analysis done by one author can be transferred to another author with little room for ambiguity. So, even though narrative-based writing may appear a long-winded process for a single author, we anticipate that the benefits it can have in collaborative writing are many. A comprehensive example that shows the use of narrative-based writing (and the corresponding tool) in a real collaborative writing scenario is presented in Chapter 7.

4.3 Applying narrative-based writing: Second example

To reinforce the steps of narrative-based writing, a second example is given below. A concise non-technical document has been chosen this time so that all three steps can be demonstrated easily and within a few pages. We do not examine any particular collaborative writing features here. In this example, we just focus on the method. We imagine the need for an author to produce a set of simple fables that teach the reader the moral lesson of being prepared for the days of necessity.

Step 1: Creating the DN

The example of the fable was something we analysed previously. The first DN we created for it was written in the old style we discussed earlier and incorporated the author's intentions and reasoning. We present this DN below just as it appeared in (Henderson and De-Silva, 2006) to serve as an example of the old type of DNs. This style has now been abandoned and a more recent version of the DN, which will be used in the rest of this example, is given too.

> I want to write a short story that will contain an implicit moral lesson. I will use animal characters with human features. I believe this will convey the wisdom in an enjoyable and memorable way. I will introduce two or three characters with opposite human characteristics (one righteous, one immoral). These characteristics will be revealed through brief conversations at the start of the story. Then there will be a series of events that will be tailored to demonstrate that the characters with the moral attitude always win and that the others suffer consequences for their unwise actions. Thus the reader will be gently persuaded to take on the characteristics of the successful characters.

Figure 4-6: Older version of the generic DN for a fable

> [There are two animal characters that have opposite human characteristics.][1] [They meet while going about their daily activities and converse.][2] [The 'bad' character enjoys momentary success and makes fun of the better character.][3] [The sequence of events after that alters this situation.][4] [The bad character is left destitute and in envy of the good character][5] [who is reaping the benefits for continuing his untiring efforts.][6]

Figure 4-7: New version of the fable DN

Step 2: Analysing the DN using RST

The next step is to analyse the DN using RST. We do not go into the details of the analysis since the process has been discussed several times already (see Chapter 3 for a tutorial on RST). A possible RS-tree for the DN is given below.

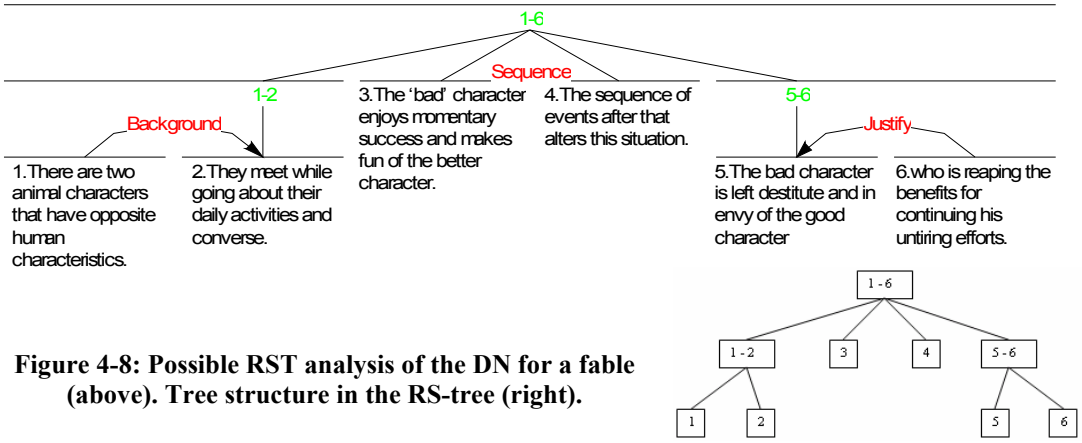


Figure 4-8: Possible RST analysis of the DN for a fable (above). Tree structure in the RS-tree (right).

Step 3: Writing the fable to fit the DN

Instead of creating a new fable, we have taken the popular story of the Ant and the Grasshopper (Long, 1997) and analysed it to show how it fits our DN. We show the segments of the DN and the corresponding sections in the fable.

1	There are two animal characters that have opposite human characteristics.	In a field one summer's day a Grasshopper was hopping about, chirping and singing to its heart's content. An Ant passed by, bearing along with great toil an ear of corn he was taking to the nest.
2	They meet while going about their daily activities and converse.	"Why not come and chat with me," said the Grasshopper, "instead of toiling and moiling in that way?" "I am helping to lay up food for the winter," said the Ant, "and recommend you to do the same."
3	The 'bad' character enjoys momentary success and makes fun of the better character.	"Why bother about winter?" said the Grasshopper; we have got plenty of food at present." But the Ant went on its way and continued its toil.
4	The sequence of events after that alters this situation.	
5	The bad character is left destitute and in envy of the good character	When the winter came the Grasshopper had no food and found itself dying of hunger,
6	who is reaping the benefits for continuing his untiring efforts.	while it saw the ants distributing every day corn and grain from the stores they had collected in the summer.

Figure 4-9: The fable of the Ant and Grasshopper structured according to the DN

4.4 Discussion

This section contains a discussion of narrative-based writing and a re-examination of the criteria we set out in Chapter 2 for a document planning technique.

A DN, on its own, is a useful guide to planning a document. Creating a DN helps authors think of the document from the reader's point of the view (i.e. the story that will eventually be transferred to the reader's mind). Also, it helps to derive a natural sequence to the ideas in the document. Doing an RST analysis of the DN adds extra information and also provides some measure of the quality of the DN. If all the segments can be linked via relationships and assembled into a tree, there is a higher likelihood of the DN being coherent.

One may also question the practice of analysing the DN as opposed to the actual document sections. The author *could* analyse the document if they wanted to. In our technique, the DN is analysed instead because the story needs to be validated before being implemented in the document. If the DN is coherent (RS-tree formed), then a document that follows that DN is

taken to be coherent too. Also, as seen earlier, not all segments in the DN will have corresponding sections in the document. Thus an analysis on the DN will have some relationships that cannot be made explicit in the document. However, these 'hidden' relationships are useful to the authors.

RST, by its very nature, helps implement some of the advice given about good writing. For instance, Gopen and Swan (1990) suggest that old (known) information should appear before new (unknown) information in a sentence. They claimed that most writers rush to record the new information and later, at their leisure, add the contextualising material that links back to the previous discourse. This burdens the reader. Mann and Thompson suggested that for some RST relationships the satellite should be presented before the nucleus (see section 3.4). The BACKGROUND relationship was one of them. Therefore, the old information (satellite) should appear before the new information (nucleus). Similarly, the segments of other relationships have recommended orders too.

Finally, we revisit the criteria from Chapter 2 that we determined were necessary for a document planning technique. We stated that a technique should:

- Support a graphical representation of ideas (like mind maps) because it provides for a better mental model of the document. The RS-tree created in narrative-based writing is a good visual aid. It shows how ideas are linked together and also shows their hierarchical ordering.
- Provide more information to justify the linear sequence of ideas (than outlines). A DN is a justification of the sequence of sections in the documents. It connects the ideas in a natural story-like fashion. We also said that some guidance as to the logical tone an author's writing should take will be beneficial. This is provided by the RST relationships.
- Be applicable to collaborative writing. By this we meant that the technique should help iron out inconsistent ideas and help the team by guiding the authors on how their sections link to others and so on. We also wanted a technique that enabled plans drawn up by one author to be transferred to and understood by another author. In our opinion, narrative-based writing achieves these goals.
- Have a way of measuring or guaranteeing the coherence of a document. By using RST, narrative-based writing allows the authors to study and verify the coherence of the DN and, therefore, the eventual document.

We conclude, therefore, that narrative-based writing is a useful aid to collaborative authors. There is some initial learning involved with regards to RST but the benefits of doing a successful analysis, we believe, outweigh this learning curve. Also, the identification of a possible subset of relationships that is applicable to technical document may help this situation even more.

4.5 Summary

In this chapter, we proposed and explained a new technique called narrative-based writing that will help authors structure a more coherent document. A summary of the steps in narrative-based writing is illustrated below[11]. Note that the dashed line indicates that an author could, if he wishes, progress directly to the writing stage after doing the DN. However, our recommendation is to complete the RST analysis before producing the document because such an analysis can benefit the coherence of the DN.

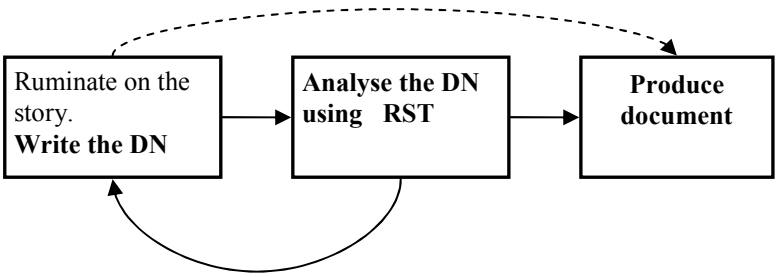


Figure 4-10: Summary of the narrative-based writing technique.

RST has, previously, been used to analyse technical documents (Rösner and Stede, 1992, Feltrim and Aluísio, 2003) but we believe the use of RST to evaluate a narrative that is subsequently used to write the document is a novel concept. Narrative-based writing formalises what most authors do subconsciously. We make use of the technique ourselves to plan the chapters in this thesis and, as evidence, present the DN for each of the chapters. The RST analyses for them are in Appendix A (section A.2).

The next step in our research is to formalise narrative-based writing and implement it in a tool that can be used by collaborative authors. This is discussed next.

Narrative-based writing is a new technique we propose for planning the structure of a document. It has three main steps: create the DN, analyse the DN using RST and produce the document accordingly. The technique is useful because it helps authors identify and improve the story of a document; thus enhancing its coherence. The new technique fulfils the gaps we recognised in the existing planning techniques.

Figure 4-11: DN for this chapter

[11] We are also working on a website which contains a tutorial on narrative-based writing (www.narratives-uk.com).

Chapter 5

A narrative-based collaborative writing tool: *The design*

Chapter 4 described narrative-based writing which is the new technique that we propose to assist with document coherence. The three main steps in the process are summarised in Figure 5-1. As discussed in the previous chapter, narrative-based writing is most beneficial to collaborative authors since a DN can help maintain a consistent story in the document. To devise a method by which coherence can be improved was the first of our goals. The second goal was to develop a tool that enabled co-authors to use this technique. This chapter begins to address this goal by presenting a comprehensive design for such a tool.

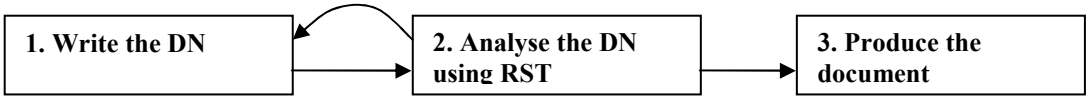


Figure 5-1: A diagram summarising the steps in narrative-based writing

As computer scientists, our interest to build such a tool was twofold. Firstly, we were curious about the effects of using the tool on technical documents. Do DNs improve coherence? Secondly, we wanted to understand and address the non-trivial issues of modelling and manipulating RS-trees. The latter, in our opinion, is the main contribution of our research.

A tool that supports collaborative work, particularly one involving complex structures such as RS-trees, requires careful design. Therefore, a graduated set of three models has been used to design this tool. Each model addresses different, progressively more refined aspects of the design that culminates in a set of formal functions. The three models described are:

1. A conceptual model
2. A business process model
3. A functional model

A simple method of version control and merging are discussed too because they are essential for collaborative writing, but are not the main focus of this research.

5.1 The Conceptual Model

5.1.1 Introduction

Sowa (1983) states that a "conceptual analysis clarifies muddled thinking and makes ideas precise." Hence, we begin the design with a conceptual model to define the key concepts. In narrative-based writing, there are three main components: the DN, the RS-tree and the eventual document. The document is not considered here since there are many other tools that support the collaborative editing of a document well. We only model the DN and the RS-tree.

5.1.2 The document narrative (DN)

A DN is a précis of the story conveyed by a document to the reader. It is divided into segments during the RST analysis[12]. Therefore, a DN can be defined as an *ordered sequence of text segments* where a segment is a string of arbitrary length. The order of the segments is important to maintain the DN intact. The DN can be changed by inserting new segments at specified positions or deleting and changing existing segments.

However, once the RST analysis is done, a DN is actually the *fringe of the RS-tree.* Consider, for instance, the RS-trees in the previous chapters. The DN could have been obtained by a pre-order traversal of the RS-tree, making it unnecessary to store *both* DNs and RS-trees in a database. **Therefore, only RS-trees will be focused on from now on.**

5.1.3 The RS-tree

On closer inspection, a RS-tree consists of two parts: the ordinary tree structure and the RST relationships. In order to model a RS-tree, both these components need to be considered.

We first study the representation of the tree structure. Let us assume that a RS-tree is made up of **nodes** and that each of these nodes has a unique identification number. Every RS-tree has a root node by which it can be identified. The leaf nodes in the RS-tree correspond to the segments in the DN. The internal nodes are spans. (A span is then a collection of adjacent nodes.)

The tree can be built using two types of relationships: "parent-child" and "next" relationships. Parent-child (PC) relationships exist between a node and its children nodes, and are typical in tree representations. It is also important, in this context, to maintain the order of the sibling nodes. This is the purpose of the Next (NXT) relationship which exists between a pair of nodes that have the same parent and follow one another. Both these types of relationships are discussed below, with reference to the RS-tree in Figure 5-2.

[12] See section 3.2 for information on how to segment a text for the RST analysis.

Parent-child (PC) relationships

A PC relationship holds between a node and its child node. It is represented using two fields: (Parent node, Child node). The PC relationships in Figure 5-2 are *(1, 2), (1, 3), (3, 4), (3, 5)* and *(3, 6)*.

Next (NXT) relationships

A NXT relationship holds between two nodes that have the same parent and are in sequence. It is also represented using two fields: (First node, Second node). In the following sections, it will become necessary to order a group of nodes according to these NXT relationships. To simplify this process, we have decided to add a special NXT relationship - (_,N) - to indicate that N is the first sibling in a set of children. Therefore, the NXT relationships in Figure 5-2 are (_,2), (2, 3), (_,4), (4, 5) and (5, 6).

Having modelled the tree structure, we move on to study the representation of the other component of a RS-tree: the RST relationships. One way of denoting a RST relationship is as a triple (e.g. node1, *motivates*, node2). These can be stored in a triple store and manipulated using the algorithms developed to retrieve information from a triple store. (See (Harris and Gibbins, 2003) for more information on triple stores.) However, we do not expect to deal with a repository of relationships so large as to warrant the use of a triple store. We stick to a simpler representation which is described below.

RST relationships

In our model, a RST relationship is stored using four fields: the relationship name, the nucleus node, the satellite node and state.

(Relationship name, Nucleus node, Satellite node, Relationship State)

The RST relationships in Figure 5-2 are: (Motivation, 3, 2, Satisfied), (Background, 5, 4, Satisfied) and (Justify, 5, 6, Satisfied).

The first three fields are self-explanatory. The ‘state’ field has been introduced to keep track of relationships as changes are made to the RS-tree. The value of this field can either be “satisfied” or “unsatisfied”. In our model, a relationship is satisfied when its nucleus and satellite (or two nuclei) fit definitions by Mann and Thompson (1988) for that relationship. A relationship remains unsatisfied until explicitly stated otherwise by the user. As authors make changes to parts of the RS-tree, certain relationships that were previously true may need to be changed to unsatisfied. The strategy used to select which relationships need to be changed is explained in the section on version control.

In multi-nuclear relationships, the second field will hold the second nucleus node. We also restrict the relationships to be binary[13]. This will affect the SEQUENCE and JOINT relationships which are allowed to involve multiple nodes. They will now have to be represented using two nodes at a time. So, if N1, N2 and N3 are in a SEQUENCE relationship, it will need to be broken down into (Sequence, N1, N2, Satisfied) and (Sequence, N2, N3, Satisfied).

These concepts are illustrated below. For clarity, the conventional tree structure is in black and the added RST information is in blue.

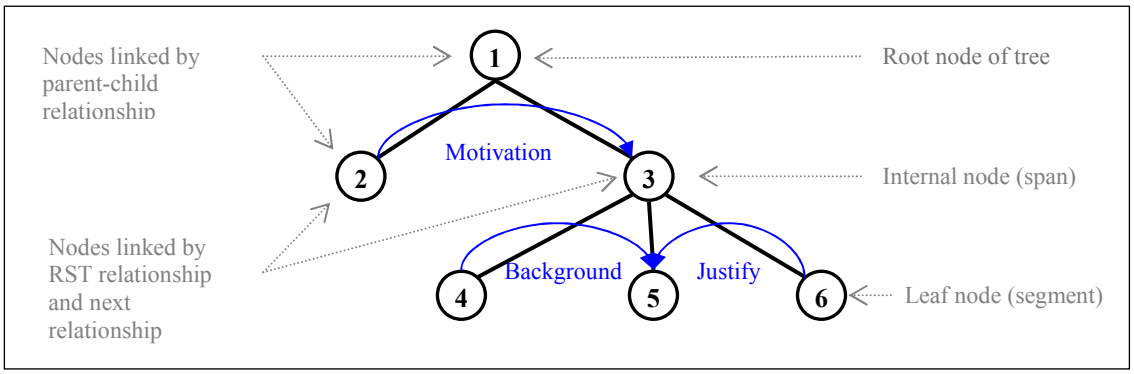


Figure 5-2: Diagram showing the components of a RS-tree

5.1.4 A sample representation

To demonstrate the use of nodes and relationships to model a RS-tree, we present an example in this section. The DN below will be used for the RST analysis.

> [There is a good initial condition] [which is disrupted by an unexpected problem.] [A solution is fast sought and executed] [to restore the initial condition.]

Figure 5-3: A sample DN

A possible RST analysis for this DN was completed and the diagram below shows how the RS-tree is stored in our model. The figure is followed by a description of its relationships and nodes.

[13] N-ary relationships can be converted to binary. Other researchers have restricted themselves to only binary RST relationships too.
MARCU, D. (2000) *The Theory and Practice of Discourse Parsing and Summarization*, The MIT Press.

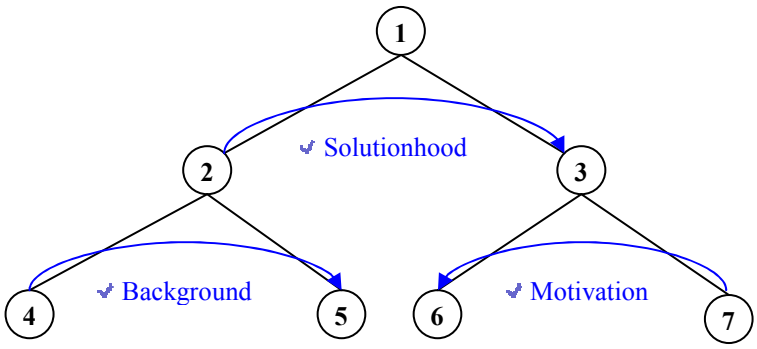


Figure 5-4: Diagram illustrating how the RS-tree for the DN in Figure 5-3 is stored using our model

The nodes necessary to represent the RS-tree using our model are listed below. Node 1 is the root node. Nodes 1, 2 and 3 are spans because they are parents of, and thereby contain, other nodes.

Node	*Represents*
1	Root node (contains nodes 2-7)
2	Span containing nodes 4 and 5
3	Span containing nodes 6 and 7
4	"There is an initial condition"
5	"which is disrupted by an unexpected problem."
6	"A solution is fast sought and executed"
7	"to restore the initial condition."

Table 5-1: Nodes needed to represent the RS-tree using our model

The relationships necessary to model the RS-tree are listed next. In Figure 5-4, a tick (✔) beside a RST relationship indicates that it is satisfied. This RS-tree will be used in the forthcoming sections to demonstrate various functions.

PC relationships:
(1,2), (1,3), (2,4), (2,5), (3,6), (3,7)

NXT relationships:
(_,2), (2 3), (_,4), (4,5), (_,6), (6,7)

RST relationships:
(Solutionhood, 3, 2, Satisfied)
(Background, 5, 4, Satisfied)
(Motivation, 6, 7, Satisfied)

Table 5-2: Relationships needed to represent the RS-tree using our model

In this chapter, RS-tree diagrams will be drawn as shown in Figure 5-4 to better illustrate the concept of nodes and the types of relationships. It differs from the RS-tree diagrams in the preceding chapters. Note that circles have been used to represent nodes (whereas rectangles were used before to denote segments). Also, even spans now have a node number. Before, they used to just contain the list of segments it represented (e.g. 3-7).

5.1.5 Summary

A DN can be obtained by taking the fringe of a RS-tree. Therefore, only RS-trees have been modelled in our design. A RS-tree can be represented completely by a set of relationships. Three types of relationships have been used for this purpose: PC, NXT and RST relationships. Each RS-tree has a root node. Meanwhile, leaf nodes correspond to segments in the DN and internal nodes correspond to spans. These concepts are summarised below.

DN	The fringe of a RS-tree
RS-tree	A set of relationships
Relationship	A PC, NXT or RST relationship between nodes
Node	A component of the RS-tree. Every RS-tree has a root node. Leaf nodes correspond to segments of the DN and internal nodes are spans.

A DN prior to being analysed using RST will be stored as a minimal tree structure (i.e. just the root node with children nodes corresponding to the segments of the DN). There will be no RST relationships. This is shown below.

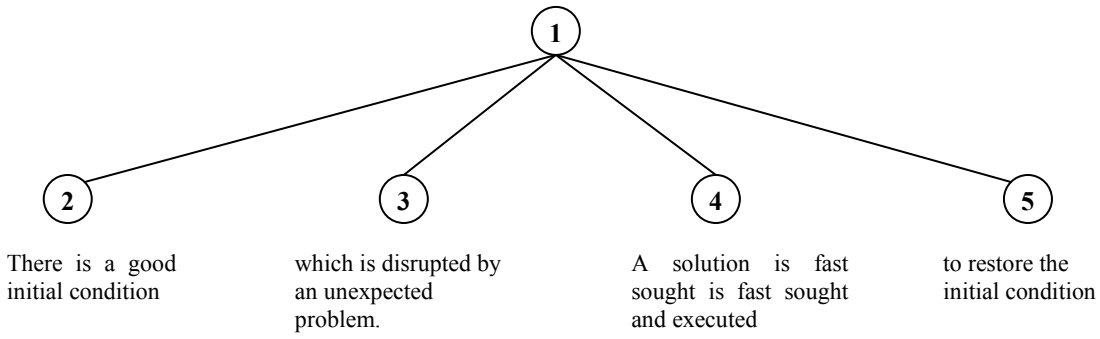


Figure 5-5: A minimal tree showing the DN before the RST analysis

The RS-trees are stored in a repository. As changes are made to a particular RS-tree, new versions of it will be created. This means that there needs to be a method to manage these versions. Therefore, version control is discussed next.

5.2 Version control

5.2.1 Introduction

A detailed study of version control is beyond the scope of this thesis. However, a tool allowing collaborative editing of RS-trees cannot be complete without some method of managing the revisions. This discussion on version control has been placed in a separate section because it was unclear as to which model it should belong to. It is necessary, however, to know the method of version control before proceeding to the business process model so that user operations can be properly specified. Hence, this was thought to be the ideal location for this section.

We present a brief overview of three version control systems existing today. The three version control systems discussed are RCS, CVS and LibreSource. RCS was chosen for historic reasons. It appears to have been one of the first tools to use the concept of *differences* (or *deltas*) to regenerate the revisions. Many of the features in RCS form the basis for the now popular tool, CVS. LibreSource is a web-based tool that allows users to work on individual replicas of a file and uses a method called *operational transformation* to ensure that the edits are applied to each replica in the right context (this was discussed in section 2.5). This overview is followed by a description of the simple method of version control designed for our tool.

5.2.2 Revision Control System (RCS)

RCS is a version control system from the 1980's. Like several other tools at the time, RCS stored just the differences between successive revisions (called deltas[14]) to conserve space. However, RCS differed from previous tools in two ways. RCS was novel in that it considered changes to the whole family of files together while previous tools had treated the components of a system in isolation. RCS also used a technique of *reverse deltas* (Tichy, 1982) to reconstruct revisions.

Files were checked in to and out of the system using the `ci` and `co` commands respectively. When a file was checked in, an appropriate version number was allocated automatically or assigned by the user. RCS also prompted for a log message that summarised the changes made. Therefore, each version contained the following information: check-in time and date, author's identification, state, log message and the actual text (only deltas). The state of a version was, by default, set to 'experimental'. The state could be changed to 'stable' or 'released' by a user. A tree of versions was created. A function called 'join' was available that could be applied to a triple of versions to merge the changes based on a common ancestor.

---

[14] When storing deltas, the grain of change is the *line*. This was because the UNIX program called `diff` computed deltas line by line.

RCS used locks to avoid conflicting changes to the same version. So, a version was locked when it was checked out by a user for editing. The lock was released once it was checked back in. These locks could be forced; thus, allowing some flexibility to the otherwise restrictive system of locking. For more information about RCS, see (Tichy, 1985).

5.2.3 Concurrent Versions System (CVS)

CVS is a free version control system that is popularly used today. It started out "as a bunch of shell scripts written by Dick Grune, posted to the newsgroup `comp.sources.unix` in the volume 6 release of July, 1986" (Cederqvist, 2002). Like RCS, it also records just the differences between the versions and is driven using CVS commands.

All the files in CVS are stored in a centralised repository. Each revision is numbered with a number of period separated decimal integers (e.g. 1.1, 1.2 and so on). Revisions on branches are numbered accordingly (e.g. 1.1.1, 1.1.2 and so on) forming a tree of versions.

Users never access these files directly. Instead, the user has to 'check out' the required file to create his own working copy of it. After the user finishes working on it, he can commit the changes to the repository, making the revised files available for anyone else using that repository. This process can be performed in one of two modes: *reserved checkouts* or *unreserved checkouts*. Reserved checkouts (or file locking) is often the only option provided in systems like RCS. It allows only one person to edit a file at a time. Unreserved checkouts, on the other hand, allow all the participants to work on independent copies of the files and CVS *merges* the changes once they are committed[15]. CVS has many other features which are explained in (Cederqvist, 2002).

5.2.4 LibreSource

LibreSource is an open-source, web-based platform dedicated to collaborative software development. Its design and development are based on the J2EE technology and an application server called 'jonas'. The LibreSource platform is available via a website and is used in a variety of applications. For example, Marjanovic et al. (2006) describe the use of LibreSource in E-learning.

Software and documents shared using LibreSource are hosted as projects. Each project is allocated a set of resources. *So6 synchroniser*, a version control system developed for LibreSource, is one of these resources (Forest, 2005). So6 is based on a technique called *operational transformation* (OT) and aims to overcome some of the shortcomings of CVS.

OT was described in Chapter 2. Users can work on copies of the shared document locally. These local workspaces are then linked to the synchroniser. The synchronisation process takes

[15] The merge works best with text files. External merge tools may need to be used for other types of files.

into account modifications done to the document by all the users and applies these to all the replicas of the document so that the users (and the server) end up with identical values.

When concurrent edits are made to the same set of lines, users are notified of conflicts. They are then required to manage and resolve the conflict. LibreSource also allows user groups and roles to be defined, along with access rights for each. This and more advanced security features are described in the LibreSource documentation (Forest, 2005).

5.2.5 Our method of version control

We are not going to implement a complex version control mechanism into our tool because that is not the main focus of our work. The method designed for the tool is simple and sufficient for the purposes of this application. Some features have been borrowed from the systems described above.

Early on in the design, it was decided that data once stored in the repository would never be changed or deleted. Therefore, any modification to a RS-tree results in the creation of a new version of that tree (as opposed to in-situ updates to the data). All unacceptable changes can be undone by reverting to a previous version.

Each new version created is allocated a unique version number - an integer – automatically. Each version also contains the name of the author that made the change and the number of the version it was derived from. We call the older version the ancestor or parent version[16]. The parent version enables users to track changes. A tree of versions is established like in RCS and CVS. This is illustrated by the diagram below where versions 3 and 5 have been derived from version 2, version 4 from 1 and so on. The first version of a RS-tree is always assigned 0.

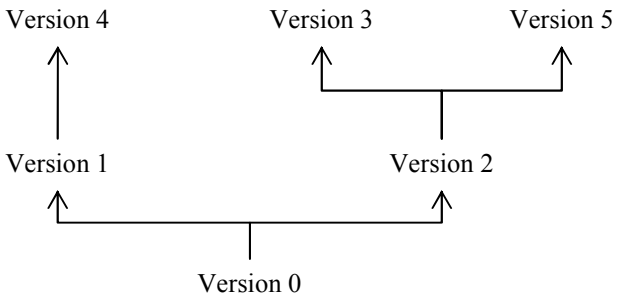


Figure 5-6: Diagram showing a tree of versions

[16] Note that the use of the word ‘parent’ in “parent version” differs to its use in the “parent-child” (i.e. parent of a node) relationships. We will distinguish clearly between the two in the following sections.

Like in RCS, each version of the RS-tree is also given a "Status" attribute which indicates whether it was created after an edit, review or a merge. The status field can, therefore, contain "Edit", "Reviewed" or "Merged."

Each time a change is made, a new version of the RS-tree is created with the change incorporated. However, only the affected nodes are replaced. This ensures that unaffected parts of the tree are not copied unnecessarily. This is similar to storing the deltas in RCS and CVS. The unchanged nodes in the parent version are linked to from the new version.

A certain process is used to determine the nodes and relationships that should be changed and those that should remain the same. Figure 5-7 is used to explain this process. It shows two versions of the RS-tree from Figure 5-4.

Node 7 in version one is changed. This node is replaced in version two and is assigned the number ten (node 10). To incorporate this replacement, node 3 has to be changed too (node 9). Node 9 is now the parent of nodes 6 and 10. Using the same reasoning, node 1 is replaced with node 8, which becomes the root node of version 2. Links are made to unaffected nodes in version 1 which are indicated by grey dashed lines.

With respect to the RST relationships, when changes are made to nodes, it is assumed that relationships involving those nodes and their parents may no longer be satisfied (and, therefore, need to be brought to the attention of the authors). Using this rule, the state of the MOTIVATION relationship in version 2 between nodes 6 and 10 is changed to "unsatisfied" due to the changes. This is denoted by a cross against the relationship name and a red arrow instead of a blue one. The Solutionhood relationship may also be affected and is set to "unsatisfied". The Background relationship remains unchanged.

Thereby, when a node in the RS-tree is changed or deleted, or a new node is added, the relationships in the *path* from the root node to the affected node are set to unsatisfied and the nodes in the path are replaced. All other relationships and nodes stay the same.

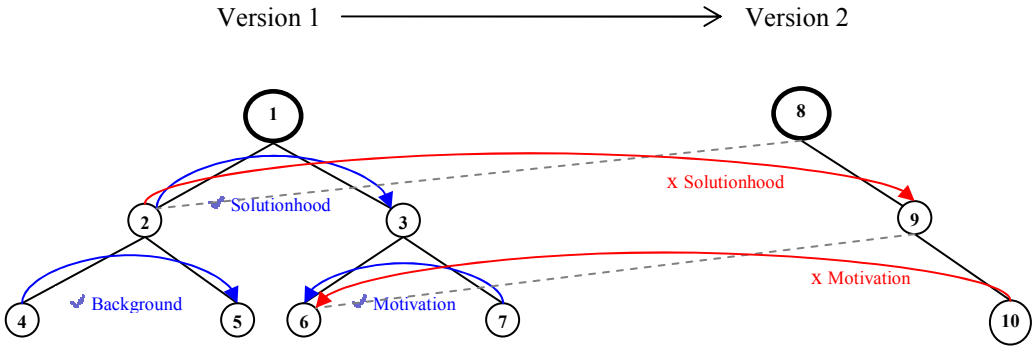


Figure 5-7: Diagram showing the creation of a new version of a RS-tree. Node 7 in version 1 is changed. Unaffected parts in version 1 are linked to from version 2.

If the order of nodes is affected by the change, some nodes that are siblings of the changed node may need to be replaced too. This is discussed in detail in the functional model. The status of both these versions of the RS-tree is "Edit". Each version will also hold the name of the author who created it. The root nodes in the versions above have been highlighted using darker lines. When multiple versions of a RS-tree exist, it is akin to a *forest* of connected RS-trees with each of them identified by a unique root node.

5.2.6 Summary

In summary, each time a user makes some change to a RS-tree, a new version of it is created and stored in the repository. However, only affected parts of the RS-tree are replaced in the new tree. The unchanged sections are just linked from the parent version.

Each version of the RS-tree contains the following information: the version number, the root node, the parent version (or ancestor), the author that created it and the status of the version. To this list, we add another attribute called 'ID'. Although the root node is a unique identifier of a RS-tree, it will not make much sense from a user's point of view. An author is likely to want to say 'I want version 2 of the RS-tree relating to a particular document'. Hence, the ID has been introduced. So, the ID together with the version number, is another way of uniquely identifying a RS-tree.

Our method of version control contains no locking. This is ideal for our application since a restrictive mechanism would not enable the authors to work naturally. A system of locks is possible future work for the tool to manage conflicting updates to the same version. For now, every update is stored in a version of the RS-tree. The merge function described at the end of this chapter allows users to merge two RS-trees derived from the same parent version.

Having studied the key concepts and a way of managing the versions, it is now possible to enumerate and define the user operations. This is done next in the business process model.

5.3 The business process model: An author's perspective

5.3.1 Introduction

It is an established fact that successful applications rely on well understood business processes (Henderson, 2000). A business process (BP) model, in this context, is an enumeration of all the actions which the authors can engage in; exactly like a use case model. It may, at first, seem unusual to describe collaborative editing as a "business". However, it has all the characteristics and complications of coordinated actions in any business. To be formal about a BP, these actions, the order in which they can be performed and the effect they have on the shared global state must be stated. This is non-trivial when there are multiple authors working asynchronously. Unusually, there appears to be no constraints on the *order* in which these actions can be performed by the author, as is typical in other business processes (Henderson and De-Silva, 2006).

In narrative-based writing, authors will perform the following basic actions: *read* a DN with its RST relationships, *edit* a DN and *analyse* the DN using RST. A fourth action – *review* - is added to this list. 'Review' is the process done at the end of the analysis to check if the RST relationships are satisfied or unsatisfied. Since only the RS-trees are stored in the repository, these actions can be restated as follows:

1. Read a RS-tree
2. Edit a RS-tree (includes editing a DN *and* analysing it)
3. Review a RS-tree

These actions are briefly described below.

5.3.2 Reading a RS-tree

To read a RS-tree, the author has to specify the ID and version number of the RS-tree he needs. Using these two values, the root node of the required RS-tree is found. Starting at this root node, the RS-tree is traversed using a pre-order traversal technique. For each node in the RS-tree, the text (if it is a segment in the DN) or an empty string (if it is a span) is displayed along with details of the corresponding RST relationships. Multiple authors can read the same version of a RS-tree at the same time.

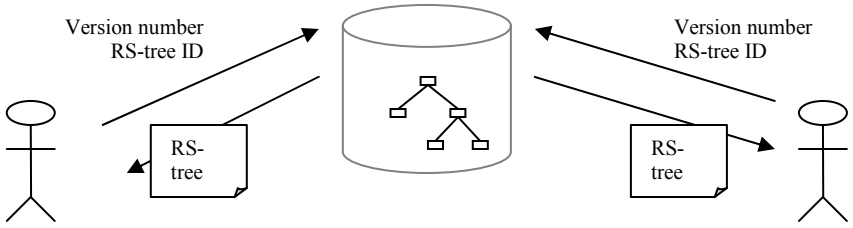


Figure 5-8: Two authors reading the same version of a RS-tree

5.3.3 Editing a RS-tree

Once again, the user needs to provide the ID and version number to identify the required RS-tree. In addition to these values, information regarding the edit (i.e. the node number being changed or removed, the new text and so on) also needs to be given. After each edit, a new version of the RS-tree is created with the affected nodes and relationships changed (as explained in section 5.2). All three types of relationships in the new version of the RS-tree – PC, NXT and RST – will need to be considered. The new version will be assigned a version number and its status will be set to "Edit".

As discussed in the section on version control, a new version is stored for every modification done to a RS-tree. This may seem like it is overkill but these data objects are tiny when compared to today's storage capacities. It also enables close tracking of activities and the ability to undo any action by going back to the previous version.

As stated in the conceptual model, a RS-tree is constituted of two components: the ordinary tree structure and the RST information. Therefore, editing a RS-tree needs to encompass changes to both these components. The corresponding user actions are described below: first the actions to edit the tree structure and then the actions to edit the RST information. Obvious validation checks such as making sure if a node exists before deleting it are not mentioned.

(1) Adding a new node at a specified position

A new node in this action represents a new segment to the DN. The position in the RS-tree where the new node has to be inserted is specified using two values: the node which should be the parent of the new node and the node after which the new node should be placed.

Either one of these values on its own is not sufficient in our model to specify an exact location. This is illustrated by a simple diagram below showing two places that a new node 6 can be inserted if the only specification given was that it should be after node 5. With both possibilities, the new node would be after node 5 when the RS-tree is traversed. (The relationship names have been left out of the diagram.)

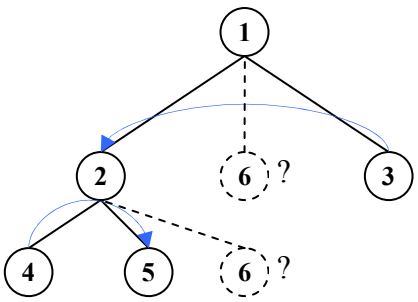


Figure 5-9: Two possible locations in the RS-tree that a new node after node 5 could be added.

This action results in a new version of the RS-tree being created with the new node added in the right location. Relationships in the path from the root node to the new node are set to 'unsatisfied' and the corresponding nodes are replaced. Since the order of the nodes is now different, the NXT relationships need to be changed too (discussed in detail in the functional model).

(2) Replacing an existing node

This action allows users to change the content of a leaf node in the RS-tree. It is equivalent to changing the text of a segment in the DN. The user needs to specify the RS-tree ID, the version number, the number of the node that needs changing and the new text. As before, a new version of the RS-tree is created with the appropriate nodes and relationships replaced. This process was illustrated in Figure 5-7.

(3) Removing a node

This action allows users to remove a node in the tree. The author has to specify the RS-tree ID, the version number and the number of the node that needs to be deleted. A new version is created with the node removed along with any relationships that involved that node.

(4) Creating a new span

The conceptual model defined a node in the RS-tree as being either a segment in the DN or a span (i.e. a collection of other nodes). Therefore, creating a span can be seen to be equivalent to adding a node. However, it is described as a separate action here since it requires different arguments and processing than the 'add a new node' action above.

To create a span, the user needs to specify a set of *adjacent* nodes with the same parent (in addition to the RS-tree ID and version number). A new RS-tree will be created with the adjacent nodes grouped into a subtree. This subtree will be a child of the nodes' previous parent node.

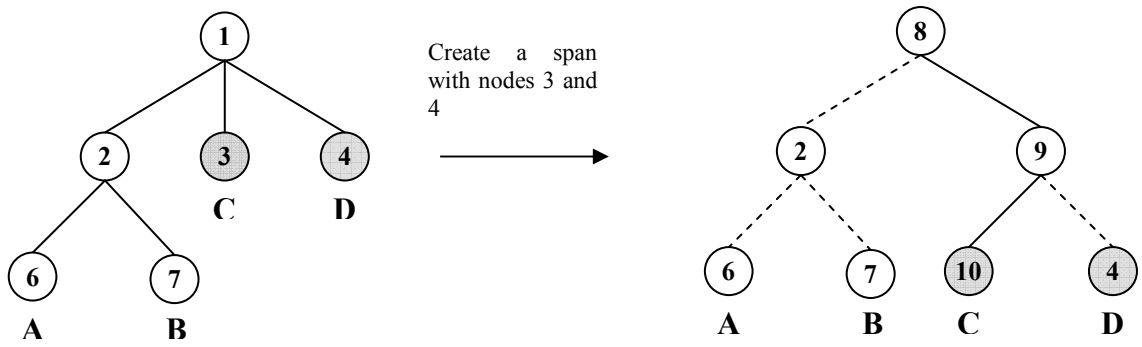


Figure 5-10: Creating a span

Spans are necessary for the RST analysis. The figure above shows a RS-tree for a DN with four segments (A,B,C and D) and a span being created with nodes 3 and 4. It is usual for these nodes to be first linked by a RST relationship. So, for example, nodes 3 and 4 will be related by a MOTIVATION relationship (say) and they will then become a span which can, in turn, become part of another RST relationship. We do not enforce this rule in our tool. So, users can, if they wish, create the tree structure (as shown in Figure 5-10) and then add the corresponding RST relationships later.

The dashed lines indicate parts of the RS-tree that have not changed. Note that the first node in the new span has been replaced with a new node (containing the same content). So, in the example above, node 3 was replaced with node 10 while node 4 remained the same. This was done in order to maintain the NXT relationships. This process is detailed in the functional model.

Next, the actions needed to edit the RST information in the RS-tree are described. The word 'relationship' below refers to RST relationships. Like in the actions above, the user needs to provide the RS-tree ID and version number. This will not be repeated in each description below.

(5) Adding a new relationship

To add a new relationship, the author needs to specify the name of the relationship and the two nodes to which it applies. In the original definition of RST (Mann and Thompson, 1988), any two nodes are only expected to have *one* relationship between them. Therefore, if the two nodes specified are siblings (i.e. have the same parent) and have no other relationship between them, a new version of the RS-tree is created with the relationship added. If there is already a relationship between the specified pair of nodes, it will be replaced by the new relationship. The state of this relationship, by default, is set to "unsatisfied" until this is changed in the review process.

(6) Editing a relationship

A relationship is modelled using four fields: the relationship name, the nucleus node, the satellite node and the state. In this action, the user is allowed to change the name of a relationship. The user has to specify the two nodes between which the relationship exists and also the change required (i.e. new name). A new version of the RS-tree is created with the relationship modified as requested. The two corresponding nodes are also replaced to accommodate this change[17]. If the user wishes to modify the nodes to which the relationship applies, then he needs to delete the relationship and add a new one.

[17] The previous relationship will continue to exist between the two nodes in the old version. In the new version, the new relationship has to then be between two similar nodes (but not the same ones).

(7) Deleting a relationship

This action results in the creation of a new RS-tree with the specified relationship removed. The relationship is specified using the numbers of the two nodes involved. (There should only be one relationship between any two nodes.)

5.3.4 Reviewing a RS-tree

Reviewing a RS-tree can be viewed as a collection of 'edit a relationship' actions where the author modifies the states of the relationships in a given RS-tree. A new version of the RS-tree is created with the states changed. In some collaborative writing teams only members who are designated as 'reviewers' are allowed to carry out this action. The status of the new version thus created is set to 'reviewed'.

5.3.5 Summary

The business process model identified the user actions necessary to perform narrative-based writing and defined each of them informally. Each action results in a new version being created. The actions are listed below.

Action performed by the author
1 Read a RS-tree
2 Edit a RS-tree
Add a new node at a specified position
Replace an existing node
Remove a node
Create a span
Add a relationship
Edit a relationship
Delete a relationship
3 Review a DN

Table 5-3: A summary of the author actions

These actions are formally defined as functions in the functional model (described next).

5.4 The functional model: An implementer's perspective

5.4.1 Introduction

At each step, the design of this tool has been refined to show more clarity and precision. The functional model in this section is the culmination of the preceding discussions. We present formal definitions for a set of functions that are necessary to implement the user operations that were described in the business process model. Some of the functions are called *core functions* because they provide basic functionality such as searching the RS-tree and retrieving the children of a given node. The other functions use a combination of these core functions to define the user actions. These functions have been implemented in Java. See Appendix B (section B.2) for the Java methods.

Descriptions of the functions below are accompanied by a sample application of the function on the RS-tree and DN from Figure 5-4. The RS-tree has been reproduced below for convenience. Note that the relationship names will be denoted by single letters to save space (S – Solutionhood, B – Background, M – Motivation).

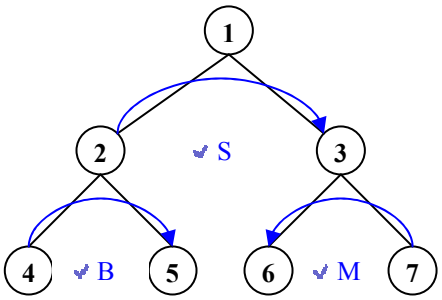


Figure 5-11: Sample RS-tree

5.4.2 The data model

The functions in this section are carried out on RS-trees in the repository. The conceptual model showed how a RS-tree can be stored using a set of nodes and relationships. The attributes of a version of a RS-tree, a node and the three types of relationships are repeated below. In addition to the data types listed below, `boolean`, `integer` and `String` have been used in the functions as well.

Attributes of a version of a RS-tree

ID	The number identifying the RS-tree
Version	Number of this version of the RS-tree
Parent version	Number of the parent version
Root node	Root node of the RS-tree
Author	Name of the author who submitted this version
Status	The status of the version (Edit, Reviewed or Merged)

Attributes of a Node

ID	The unique number identifying the node
Text	Contains the text of the node if it represents a segment. (Empty if it is a span.)

Attributes of a Parent-Child (PC) relationship

Parent	The ID of the parent node
Child	The ID of the child node

Attributes of a Next (NXT) relationship

First	The ID of the first node in the pair
Second	The ID of the second node in the pair

Attributes of a RST relationship

Name	Name of the relationship
Nucleus	The ID of the nucleus node
Satellite	The ID of the satellite node (or second nucleus)
State	'Satisfied' or 'Unsatisfied'

5.4.3 Notation used

A form of pseudocode has been used to define the functions. We have tried to make it as self-explanatory as possible but we include a few general comments below for further clarification.

- Comments are indicated by ##.
- An equals sign (=) is used for assignment and a double equals sign (= =) to test for equality.
- A 'set' refers to a collection of data elements (e.g. set(Node) is a set of Nodes) and 'add' is a function to add elements to a set (e.g. children.add(node)). Indices are used to refer to particular elements (e.g. children[1]).

- 'For' loops are defined as shown below. If more than one type of item is used in the iteration, they are separated with commas.

```
for (each item,item in collection) {
    ...action...
}
```

- Usually, 'n' is used to refer to a node and 'r' is used when the node referred to is the root of the tree under consideration. 'c' denotes the current position in the tree.
- A dot is used to refer to the attributes of an element (e.g. n.text refers to the text attribute of the node n).
- Variables are initialised as follows: `children = new set(Node)`. This initialises the variable 'children' to an empty set which can contain elements of type Node.
- STORE is an operation that stores the data in the repository.

5.4.4 The six core functions

The six core functions provide essential operations to retrieve information about a version of a RS-tree. They are: `getChildren`, `getNodes`, `contains`, `locate`, `getRSTRelationships` and `getNXTRelationships`. The results of applying each function to the sample RS-tree in Figure 5-11 are shown in the right hand panel. These core functions are used in the more complex functions in the next section.

In the functions, the particular RS-tree is indicated by specifying its root node (denoted by r). In the business process model, a RS-tree was identified using the RS-tree ID and the version number. However, this was purely for the user's benefit. Given the ID and the version number, the root node can be easily extracted.

(1) Function **getChildren(n)** examines all the PC relationships in the repository and returns the *immediate children* of node n. The child nodes are also ordered according to the NXT relationships. This function is necessary to traverse the tree. The data types of the argument and result are Node and set(Node) respectively.

```
getChildren(n){
   children = new set(Node)
   pcrelations = all PC relations
                 where n is a parent
   for (each parent,child in pcrelations){
      children.add(child)
   }
   ## order children (discussed below)
   return children
}
```

getChildren(1) = (2,3)
getChildren(3) = (6,7)
getChildren(5) = ( )

The child nodes are ordered according to the NXT relationships. There are several ways of doing this. We have used the following method:

i) Find the first child in the sequence

Extract the (_,N) NXT relationship pertaining to this set of children and find out which of the child nodes appears in N. This is the first in the sequence.

ii) Build the sequence by following the NXT relationships from the first child

In the Java, we have separated the ordering of children into a different method since it is often not necessary to order the nodes. Only a few of the forthcoming functions require the results of `getChildren()` to be ordered.

(2) Function **getNodes(r)** returns *all the nodes* in the RS-tree with root node r (including r). It makes use of the getChildren( ) function above to traverse the tree. The recursion ends when getChildren() returns an empty set[18]. The data types of the argument and result are Node and set(Node) respectively.

```
getNodes(r){
   descendants = new set(Node)
   descendants.add(r)
   children = getChildren(r)
   for(each node in children){
      descendants.add(getNodes(node))
   }
   return descendants
}
```

getNodes(1) =
(1,2,3,4,5,6,7)
getNodes(3) = (3,6,7)
getNodes(5) = (5)

(3) Function **contains(n,r)** returns `true` if the node n is contained somewhere in the RS-tree with root r. It makes use of the getNodes( ) function above. The function takes as arguments two values of type Node and returns a boolean.

```
contains(n,r){
     return (n is in getNodes(r))
}
```

contains(1,1) = `true`
contains(5,2) = `true`
contains(2,3) = `false`

(4) Function **locate (n,r)** returns the *immediate subtree* within the tree with root r that contains node n. It makes use of the getChildren( ) and contains( ) functions. The data type of the two arguments is Node and the result is an integer (the id of the root of the subtree). It returns -1 if the node is not contained in any of the immediate subtrees. This function is useful

[18] getChildren() performed on a leaf node will return an empty set.

when finding the path in a RS-tree from, say, the root node to a node that has been changed by an author.

```
locate(n,r){
     children = getChildren(r)
     for(each node in children){
         if (contains(n,node)){
             return node.ID
         }
     }
     return -1
}
```

locate(5,1) = 2
locate(3,2) = -1

(5) Function **getRSTRelationships(r)** examines all the RST relationships in the repository and returns those pertaining to the tree with root r. For each RST relationship, the function checks if both the nucleus and satellite are contained in the tree with root r. The function takes one argument of type Node and returns a set of RST relationships.

```
getRSTRelationships(r){
     rels = new set(RST Relationship)
     for(each RST relationship in repository){
       if(contains(satellite,r) AND
                          contains(nucleus,r))
            rels.add(the RST relationship)
     }
     return rels
}
```

getRSTRelationships(1) =
(Solutionhood, 3,2, Satisfied)
(Background, 5, 4, Satisfied)
(Motivation, 6, 7, Satisfied)

getRSTRelationships(3) =
(Motivation, 6, 7, Satisfied)

(6) Function **getNXTRelationships(children)** examines all the NXT relationships in the repository and returns those pertaining to the nodes in children. For each NXT relationship, the function checks if both the first and second nodes are in the set children. The exception to this is in the case of the (_,N) relationships where only N will be checked to see if it is contained in the set. The function takes one argument of type set(Node) and returns a set of NXT relationships.

```
getNXTRelationships(children){
    nxtrels = new set(NXT Relationship)
    for(each NXT Relationship in repository){
      if( (first is in children OR is _ ) AND
          (second is in children))
           nxtrels.add(the NXT relationship)
    }
    return nxtrels
}
```

getNXTRelationships((6,7))
= (_,6), (6,7)

getNXTRelationships((2,3))
= (_,2), (2,3)

5.4.5 Discussion on NXT relationships

Before proceeding to present the functions to implement the user actions, it is necessary to discuss the issues that arise when maintaining the NXT relationships. Consider, for instance, the NXT relationships pertaining to the sample tree. They are (_,2), (2,3), (_,4), (4,5), (_,6) and (6,7). Whenever a new node is added, a node is removed or a span is created, this sequence of nodes is going to be affected. When, say, a new node is added after node 4 in the sample tree, one would expect the new RS-tree below to be created (the RST relationships have not been shown). The new node is assigned the ID 10.

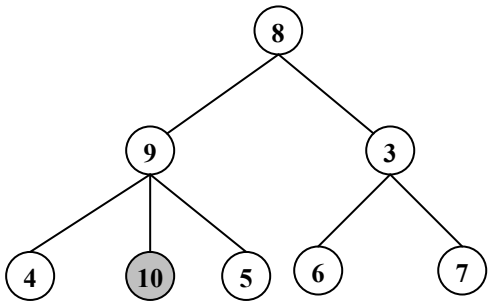


Figure 5-12: Sample RS-tree with a node added after node 4

The NXT relationships pertaining to this new RS-tree are (_,9), (9,3), (_,4), (4,10), (10,5), (_,6) and (6,7). However, when the getNXTRelationship() function is called for the children of node 9, the following NXT relationships will be returned: (_,4), (4,5), (4,10) and (10,5). As can be seen, the relationships (4,5) and (4,10) are in conflict.

To overcome this problem, we replace the node immediately after the changed node. In the example above, this would produce the tree below. Note that node 5 has now been replaced with a new node 11 (with the same text). The NXT relationships for the children of node 9 now are (_,4), (4,10), (10,11) and thus cause no confusion with the parent version of the RS-tree. A similar process has to be done when a node is removed and when a span is created.

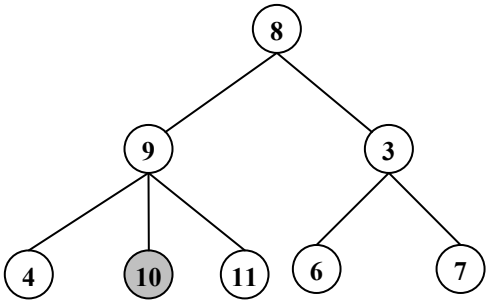


Figure 5-13: Sample RS-tree with a node added and NXT relationships restored

5.4.6 Functions to implement user actions

The following functions (which make use of the core functions above) enable the user actions of reading, editing and reviewing a RS-tree to be implemented. Once again, the results of applying each function to the RS-tree in Figure 5-11 are included.

Reading a RS-tree

This action is implemented using the function **print(c,r).** This function traverses the tree with root r and prints the details of the nodes and RST relationships. The two arguments c and r are of type Node. Argument c is used to maintain the position in the tree during the recursion.

```
print(c, r){
    display c.ID and c.text
    relationships = getRSTRelationships(r)
    for (each rel in relationships){
        if(c==rel.nucleus)
            display details of rel
    }
    children = getChildren(c)
    for (each node in children)
        print(node,r)
}
```

The results of doing `print(1,1)` on the sample RS-tree are shown below. The nodes that are spans have no associated text; hence, just the node number is displayed.

1:

2:

4: “There is an initial condition”

5: “which is disrupted by an unexpected problem”

Related to node 4 by Background

3:

Related to node 2 by Solutionhood

6: “A solution is fast sought and executed”

Related to node 7 by Motivation

7: “to restore the initial condition”

Figure 5-14: Results of doing print(1,1) on the sample RS-tree

Editing a RS-tree

The RS-tree can be edited using seven functions: four which edit the tree structure and three to edit the RST information (see business process model). These functions are listed in Table 5-4.

Functions to edit the tree structure of a RS-tree		
1	addNode(…)	Adds a new node at a specified location
2	replaceNode(…)	Replaces the text of a specified node
3	removeNode(…)	Removes a node
4	createSpan(…)	Creates a span using a specified set of adjacent nodes

Functions to edit the RST information of a RS-tree		
5	addRelationship(…)	Adds a new RST relationship between two nodes
6	replaceRelationship(…)	Changes the name of an existing RST relationship
7	removeRelationship(…)	Removes a specified RST relationship

Table 5-4: List of functions that implements the user actions

The general structure for each of these functions is:

i. Traverse the tree until the right location is found (only travelling along the path from the root to this specified position, replacing the affected nodes and relationships on the way)

ii. Once found, make the necessary change

iii. Return the root of the new RS-tree

This generic structure is tailored to perform the desired action in each of the functions. These definitions are given below.

(1) The **addNode(…)** function adds a new node at a specified location and returns the root of the new version of the RS-tree. The input parameters are below.

n: Node after which the new node should be added (left empty if new node is first)
p: Node that should be the parent of the new node
s: The text for the new node
c: The current position in the tree
r: The root of the tree

Both p and n are needed to specify the location of the new node as discussed earlier (Figure 5-9). The function traverses the RS-tree until the specified location is found, replacing affected nodes and relationships in the path. The function is outlined below.

```
addNode(n,p,s,c,r){
   1. Replicate the text of node c in a new node (say, newnode)
   2. Traverse the tree until the parent node p is found
      (a) if (c = = p)
             Store a new node with text s (say, newnode2)
             Store PC (newnode.ID, newnode2.ID)
             Store appropriate NXT relationships (see a.1 below)
      (b) else
             x   = locate (p,c)
             xx = addNode (n,p,s,x,r)
   3. Fix other relationships (see 3.1, 3.2 and 3.3 below)
   4. Return ID of newnode
}
```

Note that, by using the `locate(p,c)` function, it is possible to traverse along the path from the root to the affected location in the tree (without having to visit unnecessary nodes and subtrees). When p is found, the new node is added to the database and connected to the new RS-tree via a PC relationship.

The NXT relationships affecting the children are added too (as shown below). If n is empty, it means that the new node is expected to be the first child of node p. In this case, the NXT relationships (_,newnode2) and (newnode2, previous first child of p) have to be added. If a value has been specified for n, then the new node has to be added after n: (n.ID, newnode2.ID). If there was a node immediately after n in the original sequence, it would also have been replaced as discussed in section 5.4.5 (say, newnode3) and hence (newnode2.ID, newnode3.ID) has to be added too.

```
if(n is not empty)
   STORE NXT (n.ID, newnode2.ID)
   STORE NXT (newnode2.ID, replacement for node after n)
else
   STORE NXT (__, newnode2.ID)
   STORE NXT (newnode2.ID, first child of node p)
```
(a.1)

In step 3, all the other relationships in the path (PC, NXT and RST) are transferred to the new RS-tree. The basic idea is to change any pointers to nodes that have been replaced with their new IDs. Nearly all the functions will have a similar procedure to handle these relationships. Therefore, we discuss them at length here but will not go into as much detail in the other functions unless there is a specific point to be made.

First the PC relationships are fixed. ‘x’ and ‘xx’ are values obtained in the function above in step (2)(b). Below, if the node used to be x, it is replaced with xx. If it used to be the node after n, it is replaced with newnode3. All other relevant children are added as children of newnode.

```
children = getChildren(c)
for (each node in children){
   if (node.ID == x)
        STORE PC (newnode.ID, xx)
   else if (node is node after n)
        STORE PC (newnode.ID, newnode3.ID)
   else
        STORE PC (newnode.ID, node.ID)
}
```

(3.1)

Similarly, the NXT and RST relationships are fixed too. The states of the affected RST relationships in the path are set to “unsatisfied”.

```
nxtrels = getNxtRelationships(children)
for (each rel in nxtrels){
   if (rel.first == x)
       STORE NXT (xx.ID, rel.second)
   else if (rel.second == x)
       STORE NXT (rel.first, xx.ID)
}
```

(3.2)

```
rstrels = getRelationships(r)
for (each rel in rstrels){
   if(c==rel.nucleus){
       STORE RST (rel.name, newnode, rel.satellite, unsatisfied)
    else if (c==rel.satellite){
       STORE RST (rel.name, rel.nucleus, newnode, unsatisfied)
    ## Similarly, replace pointers to x and the node after n too
}
```

(3.3)

In the implementation stage, we will add information about the RS-tree such as its author and status into the database. For now, we concentrate on the manipulation of the nodes and relationships.

The result of doing `addNode(5, 2, 1, 1, "new text")` to the sample RS-tree is shown below. Links to nodes in the parent version are indicated using grey dashed lines. The tables following the diagram show the nodes and relationships in the repository after this change has been made. The grey information existed in the repository already (see section 5.1.4).

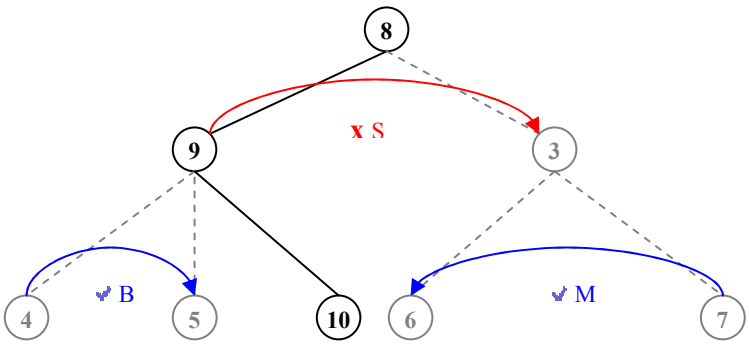


Figure 5-15: A new version of the sample RS-tree after a node is added

Node	*Represents*
1	Root node (contains nodes 2-7)
2	Span containing nodes 4 and 5
3	Span containing nodes 6 and 7
4	"There is an initial condition"
5	"which is disrupted by an unexpected problem."
6	"A solution is fast sought and executed"
7	"to restore the initial condition."
8	Root node
9	Span containing nodes 4,5 and 10
10	"new text"

PC relationships:
(1,2), (1,3), (2,4), (2,5), (3,6), (3,7)
(8,9), (8,3), (9,4), (9,5), (9,10)

NXT relationships:
(_2),(2 3), (_,4),(4,5), (_,6),(6,7)
(_,9), (9,3), (5,10)

RST relationships:
(Solutionhood, 3, 2, Satisfied)
(Background, 5, 4, Satisfied)
(Motivation, 6, 7, Satisfied)
(Solutionhood, 3, 9, Unsatisfied)

Figure 5-16: The entries in the repository showing the changes made to the RS-tree

(2) The **replaceNode(…)** function replaces the text of a specified node with new text and returns a new version of the RS-tree. This process was illustrated in Figure 5-7 in the section on version control. The input parameters are listed below.

n: Node that needs to be replaced (Node)
s: The new text (String)
c: The current position in the tree (Node)
r: The root of the tree (Node)

The function traverses the RS-tree replacing the nodes in the path and setting the affected RST relationships to 'unsatisfied'. When node n is found, it is replaced in the new tree with a node that contains text s. It is assumed that node n exists in the tree.

```
replaceNode(n,s,c,r){
     1. Generate a new node with a unique ID (say, newnode)
     2. Traverse the tree until the specified node is found
          (a) if (c = = n)
               Store newnode (containing text s)
          (b) else
               x   = locate (n,c)
               xx = replaceNode (n,s,x,r)
               Store the contents of node c in a new node (newnode)
               Fix PC relationships
     3. Fix other relationships (NXT and RST)
     4. Return ID of newnode
}
```

The process to fix the PC, NXT and RST relationships in step 3 is similar to that in the previous function and is, therefore, not repeated. The main difference is that the node immediately after n in the original sequence is not replaced. A diagram is not included either since it was illustrated in Figure 5-7.

(3) The **removeNode(…)** function removes a specified node and returns a new RS-tree. Any RST relationships involving this node are removed too. The arguments for this function are given below.

n: Node that needs to be removed
c: The current position in the tree
r: The root of the tree

```
removeNode(n,c,r){
     1. Generate a new node with a unique ID (say, newnode)
     2. Traverse the tree until the specified node is found
          (a) if (c == n)
                Replace the node just after n (newnode2)
                Set relevant NXT relationships
          (b) else
                x   = locate (n,c)
                xx = removeNode (n,x,r)
                Replicate the content of node c in newnode
     3. Fix relationships
     4. Return ID of newnode
}
```

The relationships that involve n are ignored (i.e. doing nothing to transfer them to the new tree will automatically remove them). The diagram below shows the resulting RS-tree when node 7 is removed from the sample RS-tree. As before, the links to nodes in the parent version are indicated by grey dashed lines.

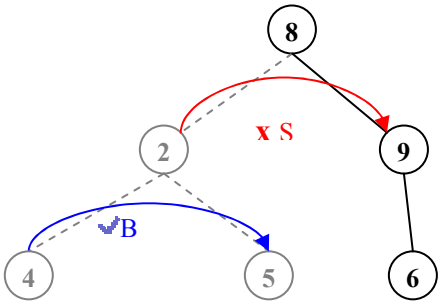


Figure 5-17: RS-tree after node 7 is removed from the sample RS-tree

In this particular case, the new NXT relationship that needs to be added is (2,9). However, if node 7 had a node immediately after it in the original version of the tree, it would have been replaced with a new node to avoid conflicting NXT relationships.

With regards to the RST relationships in the new tree, note that the MOTIVATION relationship that node 6 was involved in has been removed and the SOLUTIONHOOD relationship is set to 'unsatisfied' because it is in the path and the deletion may have affected its validity.

(4) The c**reateSpan(…)** function groups the specified set of nodes into a subtree, attaches the subtree to the nodes' common parent (p) and returns the root of the new version of the RS-tree. It is assumed that the specified nodes are adjacent and in the given tree.

nodes: Set of adjacent nodes that are to be grouped into a subtree
p: Node that the subtree will be attached to (common parent of nodes)
c: The current position in the tree
r: The root of the tree

```
createSpan(nodes,p,c,r){
      1. Store the text of node c in a new node (say, newnode)
      2. Traverse the tree until the specified parent p is found
         (a) if (c = = p)
               Create a new node (say, newnode2) ##span - no text
               Store PC (newnode.ID, newnode2.ID)
               Store relevant NXT relationships
               for (each node in nodes)
                  Store a new node for the first node in the span
                  For rest, store PC (newnode2.ID, node.ID)
         (b) else
               x   = locate (nodes[1],c) ##Since nodes are siblings
               xx  = createSpan (nodes,p,x,r)
      3. Fix relationships (discussed below)
      4. Return ID of newnode
}
```

A new RS-tree is used to demonstrate this function. The previous sample tree, while being ideal for the other functions, would not have demonstrated `createSpan()` well since the nodes in it have a maximum of just two children.

The result of applying `createSpan ((3,4), 1, 1, 1)` to the RS-tree below is shown. A new span (node 9) is created with nodes 3 and 4 as its children. Node 9 can then be used in RST relationships with either node 11 or node 2.

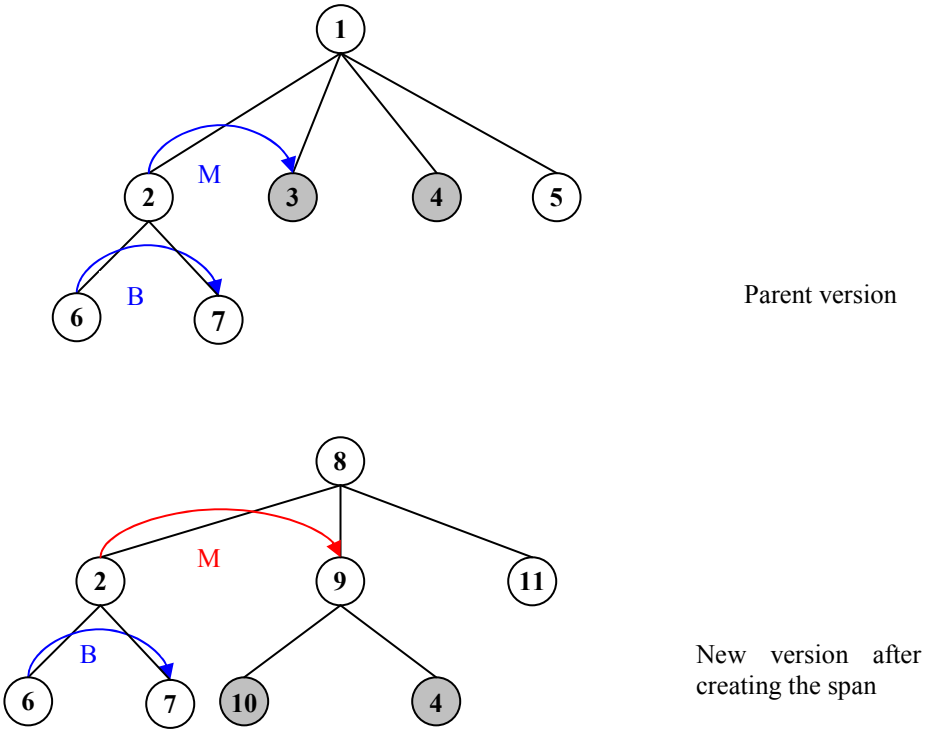


Figure 5-18: The resulting RS-tree (below) after createSpan ( ) was applied to the tree above.

In our opinion, RST relationships that involved the nodes in the span can now be applied to the span. Their state is "unsatisfied" since it needs to be checked if these relationships are still applicable. In the example, the MOTIVATION (M) relationship now exists between nodes 2 and 9. The PC relationships are also relatively trivial to fix. Once again, we discuss the NXT relationships in detail.

The nodes 3 and 4 are already in a NXT relationship from the parent version. The new information that has to be added is that node 3 is now also the first node in the sequence (which once again could be problematic since a (_,3) relationship will seem relevant to the parent version too even though it is not). Our solution has been to replace the first node in the span with a new node; thus requiring new, unambiguous NXT relationships. Note that node 5 has been replaced with node 11. This was also done to avoid conflicting NXT relationships between the children of node 8.

(5) We now go on to the functions that edit the RST information in the tree. The **addRelationship(…)** function inserts a specified RST relationship between two nodes and returns the root of the new tree. The arguments for the function are as follows:

n1:	Node which will be the nucleus
n2:	Node which will be the satellite (or second nucleus)
rel:	The name of the relationship
r:	The root of the tree
c:	The current position in the tree

The arguments are all of type Node, except 'rel' which is a string. The function assumes that n1 and n2 are sibling nodes in the tree and that there is no other relationship between them. The new relationship is set to 'unsatisfied'.

```
addRelationship(n1,n2,rel,r,c){
 1. Store the text of node c in a new node (say, newnode)
 2. Traverse the tree until the specified nodes are found
     (a) if (c!=n1 AND c!=n2)
            x1 = locate (n1, c)
            x2 = locate (n2, c)
            xx1 = addRelationship(n1,n2,rel,r,x1)
            if (x1 != x2){
                xx2 = addRelationship(n1,n2,rel,r,x2)
                STORE new RST relationship between xx1 and xx2
                STORE new NXT relationship between xx1 and xx2
                Fix PC relationships affected by xx1 and xx2

 3. Fix other relationships (similar to previous functions)
 4. Return ID of newnode
}
```

Once again, the `locate(…)` function is used to find the path to the nodes n1 and n2. Since n1 and n2 are assumed to be siblings (i.e. have the same parent), `locate(…)` should keep returning the same values. If `locate(n1,c)` and `locate(n2,c)` return different values, it means that the relevant nodes have been reached. This check is used to determine when to add the new relationship. The relationship is, by default, set to be "unsatisfied" (until explicitly changed later in the review stage).

The other relationships are processed in much the same way as it was done in the `addNode(..)` function. Pointers to nodes that have been replaced in the new RS-tree are corrected.

The sample RS-tree that has been used to demonstrate the previous functions has a RST relationship between all possible adjacent nodes already. Therefore, another example is used instead.

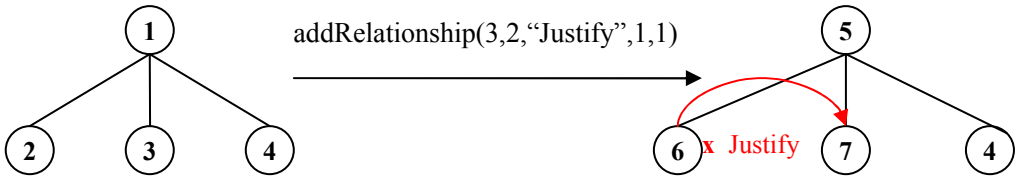


Figure 5-19: RS-trees showing the application of the addRelationship function

(6) The **replaceRelationship(…)** function replaces the specified relationship and returns the root of the new tree. This function can be used to change the name of an existing relationship. It has a similar structure to the function above. The arguments are listed below.

n1:	The nucleus
n2:	The satellite (or second nucleus)
rel:	The new name of the relationship
r:	The root of the tree
c:	The current position in the tree

```
replaceRelationship(n1,n2,rel,r,c){
 1. Store the text of node c in a new node (say, newnode)
 2. Traverse the tree until the specified nodes are found
    (a) if (c!=n1 AND c!=n2)
            x1 = locate (n1, c)
            x2 = locate (n2, c)
            xx1 = replaceRelationship(n1,n2,rel,r,x1)
            if (x1 != x2){
                xx2 = replaceRelationship(n1,n2,rel,r,x2)
                STORE new RST relationship between xx1 and xx2
                STORE new NXT relationship between xx1 and xx2
                Fix PC relationships affected by xx1 and xx2

 3. Fix other relationships (similar to previous functions)
 4. Return ID of newnode
}
```

It needs to be noted that this function is identical to the addRelationship() function above. In the Java, they have been implemented using one method. However, in the design, we keep them as two functions because, in principle, they are two different processes.

(7) The **removeRelationship(…)** function removes the specified relationship and returns the root of the new RS-tree. The arguments, all of type Node, are:

n1: Node which is the nucleus of the relationship to be removed
n2: Node which is the satellite (or second nucleus)
r: The root of the tree
c: The current position in the tree

```
removeRelationship(n1,n2,r,c){
 1. Store the text of node c in a new node (say, newnode)
 2. Traverse the tree until the specified nodes are found
      (a) if (c!=n1 AND c!=n2)
            x1 = locate (n1, c)
            x2 = locate (n2, c)
            xx1 = removeRelationship(n1,n2,r,x1)
            if (x1 != x2){
                xx2 = removeRelationship(n1,n2,r,x2)
                Fix PC and NXT relationships involving x1 and x2

 3. Fix other relationships
 4. Return ID of newnode
}
```

The name of the relationship is not specified since there should only be one relationship between n1 and n2. Once again, the tree is traversed using the `locate(…)` function. All the RST relationships in the parent tree are added to the new tree, except the one being removed.

Reviewing a RS-tree

Reviewing a version of a RS-tree involves changing the state of several RST relationships. This function takes a set of RST relationships and incorporates them all into a new RS-tree. The function is not discussed as length because of its similarity to the `replaceRelationship(…)` function.

5.4.7 Summary

This section presented formal definitions for the functions necessary to implement the user actions discussed in the business process model. Six core functions were identified that provided basic functionality. These core functions were then used to define the bigger functions that corresponded to the user operations of reading, editing and reviewing a RS-tree. The functions have been implemented in Java (see Appendix B).

5.5 Merging

Since co-authors can create divergent versions of the same parent RS-tree, it is useful to be able to merge these changes. Therefore, a simple merge function has been designed that takes two versions of a RS-tree and produces one RS-tree. The merge function is being discussed separately from the functional model because it is *extra* functionality (not part of our business process model) that was considered essential for collaborative working.

Both trees are traversed simultaneously. The merged tree will be produced using the following rules:

- If two nodes are identical (i.e. same IDs *or* same text) at the same level in the tree, then include one instance of this node in the merged tree.
- If there are two identical relationships between the same pair of nodes in both trees, add one instance of the relationship.
- Include all non-identical relationships and nodes. This may mean that a pair of nodes in the merged tree may have more than one relationship between them. This is acceptable in this situation and the authors are left to choose the most appropriate relationship.

This algorithm works best if the two versions are derived from the same parent or if one is derived from the other. The diagrams below are used to better explain this process. Version 2 of the RS-tree is derived from version 1 after changing the text of node 6. Version 3 is the merged tree. Both nodes 6 and 9 (and corresponding relationships) are included in version 3 because they are not identical.

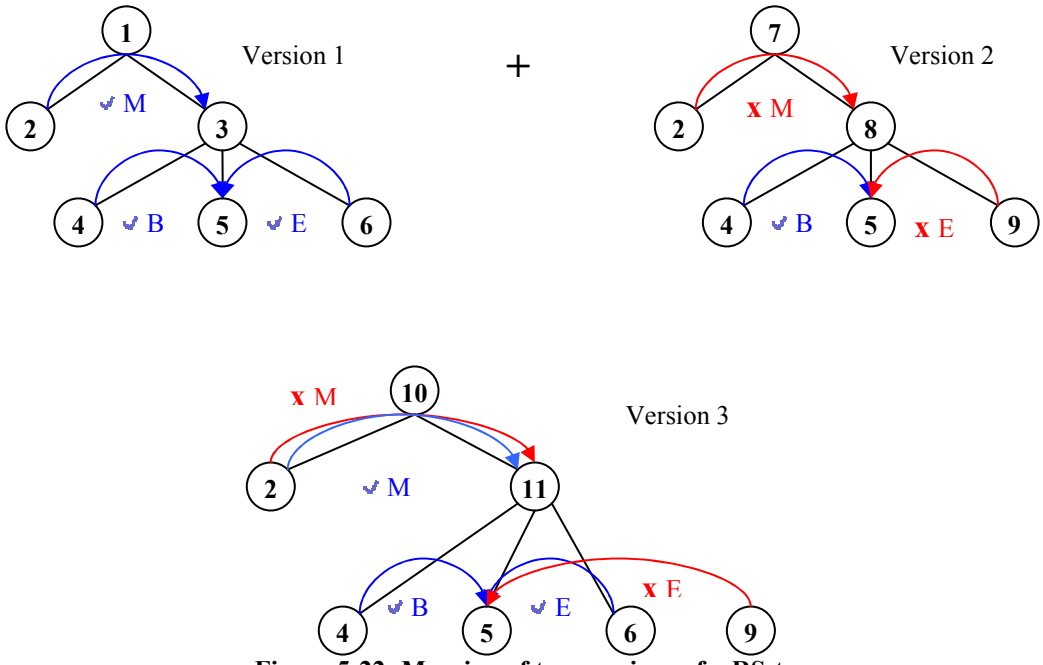


Figure 5-22: Merging of two versions of a RS-tree

5.6 Summary

Having described the narrative-based technique in Chapter 4, this chapter begins to address the second goal of our research: to develop a tool that allows collaborative authors to engage in narrative-based writing.

This chapter presented the design for our tool using three models: a conceptual model, a business process model and a functional model. The concepts in narrative-based writing such as a DN, RS-tree, node and relationship were clarified using the conceptual model. Using these concepts, the BP model described the actions that an author would expect to perform on the RS-trees in the repository. These operations were broadly categorised into reading, editing and reviewing a RS-tree. The functions necessary to implement these actions were formally defined in the functional model. As with many collaborative working tools, version control and merging had to be addressed too. We have devised simple methods to maintain the versions and merge two versions derived from the same parent. Although these methods are simple in comparison to the technologies that exist today, they are adequate for our tool.

The next chapter discusses the implementation of our tool.

A tool that is expected to support collaborative editing, particularly of non-trivial structures such as RS-trees, needs careful and thorough design. Therefore, the design for this tool has been done using a graduated set of three models. A conceptual model defines the main concepts of narrative-based writing. A business process model identifies a set of user actions which are then defined formally in the functional model. Methods for version control and merging have been designed as well since they are essential for collaborative editing (even though they are not the main focus of the tool). These functions will now be implemented.

Figure 5-23: DN for this chapter

Chapter 6

A narrative-based collaborative writing tool: *An implementation*

The previous chapter presented the design of a tool that enabled teams of authors to engage in narrative-based writing. The design was divided into three main sections: the conceptual model, the business process model and the functional model. This chapter describes an implementation that is a proof of concept of this design.

We realise that there are several technologies that could have been used and different ways in which these could have been combined. We present one possible implementation. Our choices of technology are HTML and JSP for the user interface, Java for the functions and a relational database (RDB) to store the RS-trees. Figure 6-1 illustrates this three-tier architecture.

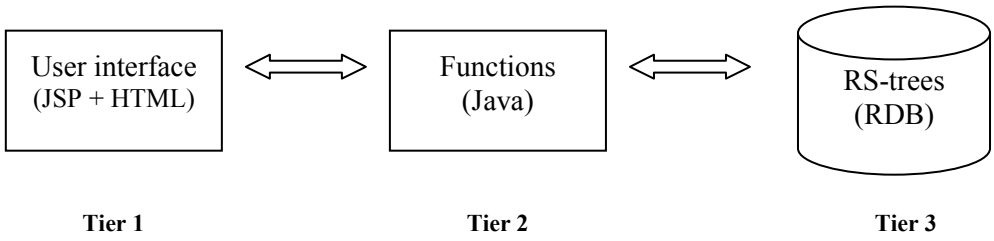


Figure 6-1: The three-tier architecture of the tool

The implementation of each of the tiers is discussed in this chapter (starting with the third). For each, we also compare some of the alternative technologies and highlight points specific to RS-trees and collaborative editing.

Two prototypes were developed prior to this tool to experiment with various ways of storing RS-trees and supporting narrative structures in technical documents. These are described in Appendix B. We refer to these prototypes briefly in this chapter when discussing alternative technologies such as XML databases.

6.1 Tier three: Database

The implementation of the RDB is presented first. In collaborative editing, documents can either be held in a central repository (to which changes are submitted) or replicated in local workspaces. We compare these two architectures below. One might also question the use of a relational database for storing tree structures instead of XML. Therefore, the use of XML to store RS-trees is discussed too.

6.1.1 Distributed vs. a centralised document repository

There are different ways that a database can be replicated. One way is to replicate the *entire* database in multiple locations. This has major complications when it comes to keeping the data up-to-date. The other way is to replicate *parts* of the database so that they are nearer the users maintaining it. The subject of this discussion is the replication of the *RS-trees*. The RS-trees could be copied in each of the authors' sites. Each author can edit his/her personal copy and 'submit' the changes. The copies will then be merged and the conflicting changes reconciled. We refer to this as a distributed document architecture and is done, for example, in LibreSource (Forest, 2005).

We have chosen a centralised document architecture for our tool instead. While the distributed architecture above is an effective way of collaborative working, the merging techniques needed to continuously monitor ongoing changes were beyond the scope of our research. It is also anticipated that, in the future, this narrative-based tool can be integrated into existing software that already has established merging mechanisms (see Chapter 9) making it unnecessary to focus on them here. In our tool, users modify versions of the RS-trees that are all held in one central database. The changes are submitted to the system and new versions of the RS-trees are stored in the repository.

6.1.2 XML vs. relational databases

XML is a natural way to store hierarchical data structures, particularly ones where there are varying amounts of text in the nodes. The RSTTool (O'Donnell, 2000) that has been used to draw some of the RS-trees in this thesis also stores the structures in XML files. We too used XML in the previous prototypes (see Appendix B, section B.1). In particular, URML (Underspecified Rhetorical Markup Language) which is an XML format for storing RS-trees (Reitter and Stede, 2003b) was used in the second prototype.

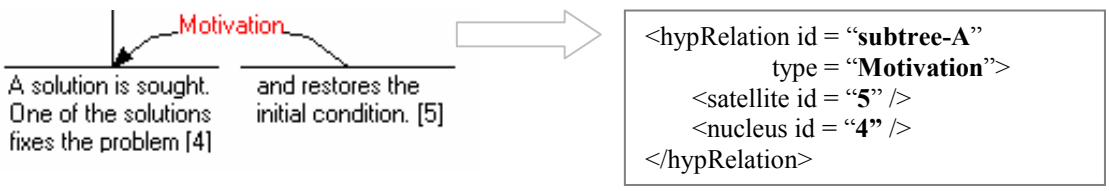


Figure 6-2: Storing a MOTIVATION relationship in URML

URML was introduced by Reitter and Stede to enable "underspecified" (or incomplete) RS-trees to be stored. This is harder with natural XML since it is common to have the entire tree defined at the start. This is not practical with RST analyses. URML bridged this gap by defining identifiers for the subtrees in a RS-tree. For example, above, "subtree-A" is an identifier for the span created by the MOTIVATION relationship.

Traversing the XML using Java was non-trivial. So, in the previous prototype, we used Xindice, a native XML database system (Xindice, 2004), to store and manage the XML files. This simplified the querying and updating of the RS-trees.

We have not used XML for this tool in order to explore the use of relational databases. Relational databases are a closer match to the data model in Chapter 5. They are well established and have several features (such as locking records and quick access using unique indexes) that would enable us to easily programme asynchronous editing without interference. However, it needs to be stressed that each of these technologies has individual strengths and trying both have been valuable experiments. Another option for a future implementation of this tool will perhaps be a combination of XML and a RDB.

6.1.3 Developing the relational database

The first step in developing the database was identifying the necessary tables and their fields. From the descriptions in Chapter 5, a table was necessary to store the details of:

1. Nodes in the RS-trees (NODE)
2. RST relationships (RSTREL)
3. Parent-Child relationships (PCREL)
4. Next relationships (NXTREL)
5. The versions of the RS-Trees (RSTREE)

Some normalisation was done to these tables. In the above state, the name of the RST relationship would be stored in each record in the table RSTREL. Therefore, a sixth table called RELATION was introduced to store the names of the 23 RST relationships from Mann and Thompson. The identifier for each relationship is then used in the RSTREL table instead of the name.

The fields of each of the tables are listed. The primary key(s) of each are indicated by an asterisk.

(1) TABLE: NODE

This table contains the ID and text of all the nodes in the RS-trees. If the node is a leaf node, the text field will contain the text in that node. If the node is a span (internal node), the text field will be empty.

Row name	Data type	Description
ID*	Number	Unique identifier
Text	Text	Text of the node

(2) TABLE: RSTREL

This table has the details of all the RST relationships. Node_1 and Node_2, together, are used as the primary key (since only one RST relationship can exist between any two given nodes). The state of the relationship is a boolean field. If the relationship is satisfied, it will be true. If not, it will be false.

Row name	Data type	Description
Node_1*	Number	Nucleus
Node_2*	Number	Satellite (or second nucleus)
Relation_ID	Number	ID of the relationship
State	Boolean	Satisfied (true) or unsatisfied (false)

(3) TABLE: PCREL

This table has all the PC relationships.

Row name	Data type	Description
Parent	Number	Parent node
Child	Number	Child node

(4) TABLE: NXTREL

This table has all the NXT relationships.

Row name	Data type	Description
First	Number	First of the two nodes
Second	Number	Second of the two nodes

(5) TABLE: RSTREE

Each row of this table contains the details of a version of a RS-tree. Most of the fields below were introduced in Chapter 5. A new ‘title’ field has been added to store the name of the document that the RS-tree corresponds to (e.g. “Paper for ICEIS conference”). This again was to make it more user-friendly.

If this table were to be completely normalised, the title and author fields need to be removed from here and replaced with corresponding IDs (like Relation_ID in RSTREL table). Two separate tables called AUTHOR (Author ID, Author name) and DOCUMENT (Document ID, Document name) will be necessary. However, we do not include these extra tables for now to keep the database simple.

As mentioned in Chapter 5, the ID of the RS-tree together with its version number form the primary key. (The root node too could be used as the primary key of this table.)

Row name	Data type	Description
ID*	Number	ID of the RS-tree
Version*	Number	Version number
Title	Text	Title of the document (E.g. "Thesis")
Root_node	Number	Number of the root node
Parent_version	Number	Number of the parent version
Status	Text	Edit, Review or Merged
Author	Text	Name of the author

(6) TABLE: RELATION

This table contains the names of the 23 RST relationships from the Mann and Thompson paper. These IDs are used in the RSTREL table.

Row name	Data type	Description
ID*	Number	ID of the RST relationship
Name	Text	Name of the RST relationship

These tables were stored using **Microsoft Access.** In addition to the tables above, two tables were added for "housekeeping" purposes. Since unique identifiers needed to be generated for the new nodes and versions of RS-trees, tables called INDEX and INDEX2 were added to store the latest identifiers for the relevant tables. When new items are added, the values in these tables are incremented to generate new IDs. We have decided to do this instead of using the 'autoincrement' feature in Microsoft Access in order to have fine-grained control over the IDs.

6.2 Tier two: Functions

6.2.1 Functional programming languages vs. Java

Since the functions in Chapter 5 deal with trees and use recursion, it may seem more usual to use a functional programming language. Functional programming enables the activities on a tree to be decomposed into smaller, reusable functions and "glued" together (Hughes, 1989).

Recursion, in general, is not preferred. It can even be slower in functional programming. However, with processor speeds of today, the performance time is rarely a factor that needs to be considered for an application like this. Recursion also makes use of a stack. The stack size is not just limited by the memory size but some compilers dictate a stack size as well. It is relatively easy to reach this limit. However, we do not anticipate there being so many calls that will break the stack size.

Java was selected to build the functions for this tool instead of a functional programming language mainly because of our previous experience in it. We were keen to have a prototype of the tool soon and using a familiar language was the best way forward. We have made use of some functional programming concepts such as breaking the functionality down into smaller, general methods that can be reused.

6.2.2 Implementing the functions in Java

The second step in the development process was implementing the defined functions. To match the design closely, classes to represent a node and each of the relationship types seem necessary. However, this has not been done in the implementation. Integers have been used instead to identify the nodes and the relevant information has been extracted from the database. So, for example, the `getChildren(n)` function in Chapter 5 that took a Node as argument, takes an integer in the actual Java. One motivation for this was to have complete control over the shared entities. This was considered necessary for more complex collaborative editing.

Each version of a RS-tree contains the name of the author that created it. In the functions in Chapter 5, we omitted the author's name as an argument to avoid overcrowding the functions. The Java methods take the author name as an argument.

Apart from these differences, there is a one-to-one correspondence between the Java methods and the pseudocode outlined in Chapter 5. Some additional methods were necessary such as methods to generate new IDs and retrieve information from the database. See Appendix B (section B.2) for a listing of the Java. The validation of the input data (e.g. such as checking if a node exists in a tree) is not discussed since it is trivial. The Java methods are based on the functions in the design and have been tested. With regards to our application, this is enough verification that the functions are correct.

There are some issues, particular to collaborative writing, that need to be addressed by our tool such as version management, interference and author authentication. Version management was discussed in Chapter 5 and the implementation has adhered to this design.

In the tool, every change made by an author will be stored as a separate version and thus, the problem of lost updates is unlikely to arise. More than one author could decide to edit version 3 (say) of a RS-tree. Each of them will submit changes. These changes will not interfere because they will be stored as two separate versions derived from the same parent version. In the remote possibility that each author submits the changes at the *exact same time*, there is a chance that the version number will not be incremented properly. However, this event is so rare that we have not studied this aspect in great detail. The use of the Java `synchronize` (Friesen, 2004) may be a possible way of making the critical parts of the functions more safe.

There is currently no security implemented in the tool because any change done by an author can be 'undone' by reverting to an older version. However, for a more professional tool that dealt with sensitive documents, security would need to be considered. For the purposes of this research on document coherence, it is not essential.

6.3 Tier one: User interface

6.3.1 Standalone vs. web-based applications

Several collaborative working tools are standalone applications (e.g. CVS). The advantage with these is that languages used to build standalone applications generally have features to design better user interfaces. However, standalone applications make maintenance harder since any change needs to be replicated in each author's copy of the software. Authors may also be reluctant to spend time downloading, installing and learning the new application.

Web-based applications, on the other hand, are more versatile, contemporary and easier to build and use. For instance, the HTML and JSP used in the current tool is comparatively trivial and took a very short time to create. The server-client architecture in Web-based applications also makes modifications easier and almost transparent to the users. For these reasons, a web-based interface was chosen for our tool.

6.3.2 Implementing a Web-based interface

The interface to the tool is implemented by a set of JSP pages. The information is gathered from the user, validated and sent to the relevant Java method. The results of the Java method are then displayed using HTML. The tool can be accessed by pointing a web browser at the specified URL. The HTML pages were tested on several major web browsers to make sure it rendered properly in all of them.

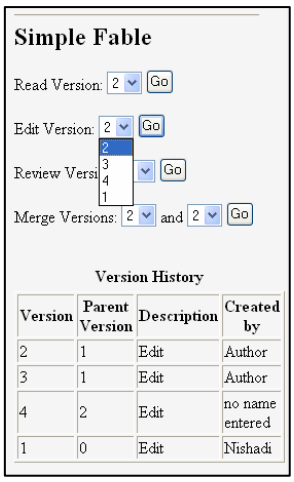


Figure 6-3: Menu

The main page is divided into three frames. Even though there have been debates about the use of frames, we found them useful to place content in independent panels. The same effect can be achieved using JavaScript. However, the frames version was simpler.

The left frame contains the menu. The menu enables a user to select an existing RS-tree or create a new one. Once a RS-tree has been chosen, a second menu (Figure 6-3) allows the user to specify the version that he/she wants to work on from a drop down list of the available versions. The drop down list makes it easier for the author and also guards against invalid user entries. A table at the bottom of the left frame displays a history of the versions for the RS-tree. It displays the version number, the parent version, its status and the author who worked on it.

The top-right frame (Figure 6-4) displays the required version of the RS-tree along with options to edit, analyse or review it. This frame reverts to the latest version of the current RS-tree if an alternative is not specified. When a RS-tree is displayed, relationships that are satisfied are displayed in blue and those that are unsatisfied in red so that authors are immediately aware of sections of the DN that may need attention.

The textual representation of the tree (as shown in Figure 6-4) may not be ideal to visualise the RST structure. It is also in a different orientation from the RS-trees normally seen and drawn (even though the RSTTool also provides an option to view RS-trees this way). However, time did not allow for us to build a graphical interface.

The bottom-right frame allows the author to read another version of the RS-tree at the same time. This helps make comparisons. There is also a Help document for users wanting more information about how to use the tool

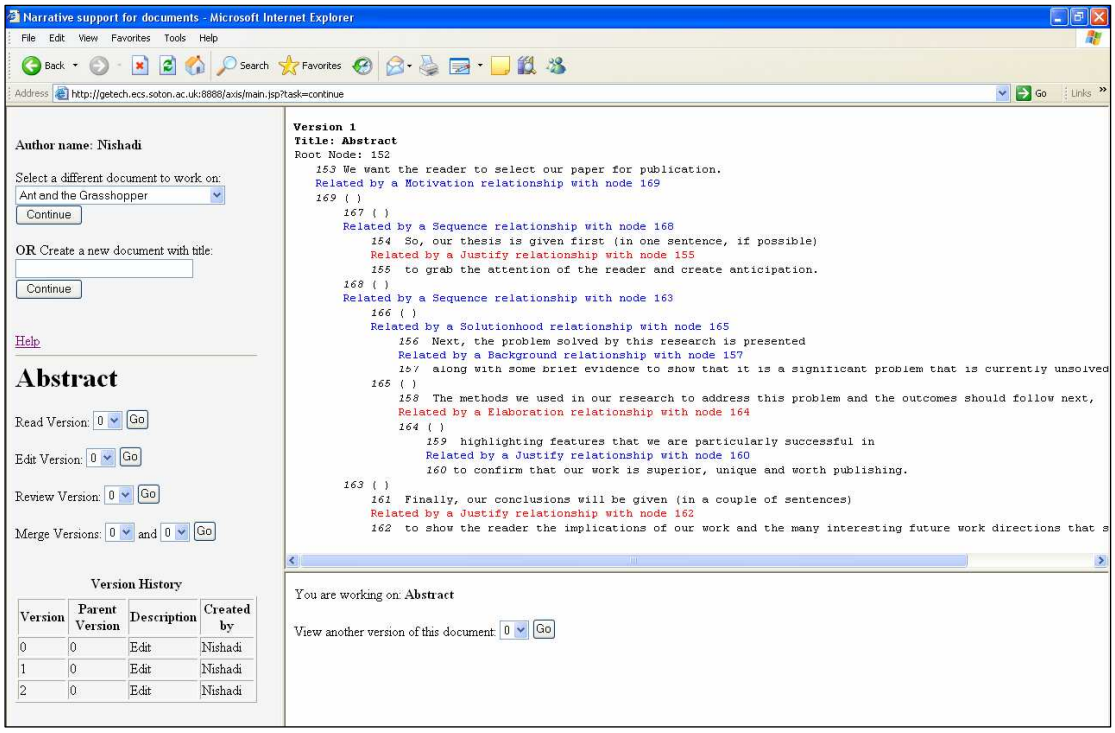


Figure 6-4: Screen shot of tool

6.4 Summary

This chapter discussed the three tiers in the implementation of our tool. This tool is a proof of concept of the design that was presented in the previous chapter.

The tool is implemented using Java, JSP, HTML and a relational database (maintained in Microsoft Access). The Java methods, listed in Appendix B, correspond directly to the functions defined in the design; thus our implementation matches the specification. The Java methods were tested but this testing is not discussed in this chapter since it is not relevant to this evaluation. We were able to build this implementation relatively quickly owing to the good, disciplined design. The functions in Chapter 5 are taken to be correct because we have implemented them in Java which has been tested.

This chapter marks the end of the design and development of the tool. The next chapter (Chapter 7) shows how the narrative-based technique and the tool can be used in a variety of technical writing scenarios. The following chapter (Chapter 8) evaluates the technique and tool.

Different implementations can be done based on the design presented in Chapter 5. We present one possible implementation. We are aware that there are other technologies and architectures that could have been used. However, the choices made were justified for the goals we wanted to achieve at the time of implementation. The tool has all the functionality described in the design including features to deal with interference and version control.

Figure 6-5: The DN for this chapter

Chapter 7

Case Studies

To recap: Chapter 4 introduced a new technique for structuring documents called narrative-based writing. Chapters 5 and 6 described the design and implementation of a tool that allowed a team of authors to use this technique to plan their document. It is now necessary to:

a) Show how the technique and tool can be used
b) Evaluate the technique and tool

This chapter shows how narrative-based writing can be used by presenting four case studies. The next chapter will contain an evaluation by way of an experiment involving some volunteers, a critical appraisal of our tool and a re-examination of our initial goals.

We propose that the narrative-based technique and tool can be applied to a variety of technical documents; with particular benefits in collaborative writing. It is not restricted to written documents however. The technique can be extended to presentations and websites, too. Therefore, this chapter presents the use of narrative-based writing in a collaborative writing scenario, highlighting the communication between the authors and the ways in which the document changes according to the evolving DN. This is followed by applications of narrative-based writing on a research proposal, a conference presentation and a website to demonstrate that the technique is suitable for a variety of forms of technical communication.

7.1 The impact of DNs in collaborative writing

This section presents an example showing how the narrative-based tool and technique can be used to plan a document produced by multiple authors. The example is a rational reconstruction of the process by which Hala Skaf-Molli and I wrote our joint paper. We did not meet face-to-face to plan it and a lot of the structure was determined by exchanging DNs at the start. A similar example also appeared in that paper (De-Silva and Skaf-Molli, 2006). A fictional third author has, however, been introduced here to make the writing task more complex. Apart from that, the example has been kept deliberately small so that the necessary aspects of collaboration can be demonstrated easily.

Let us imagine three authors (A, B and C), not in the same location, with the task of writing a joint paper about their research on merging algorithms and narrative-based writing. Authors A and B are authorities on merging algorithms while Author C is involved in narrative-based writing. They hope to divide the sections of the document according to their expertise.

To get the ball rolling, Author A comes up with a DN for the paper. He inputs the DN into the tool and does a RST analysis of it. Both the DN and RST analysis now become available to the other authors (version 1).

VERSION 1 (by Author A)

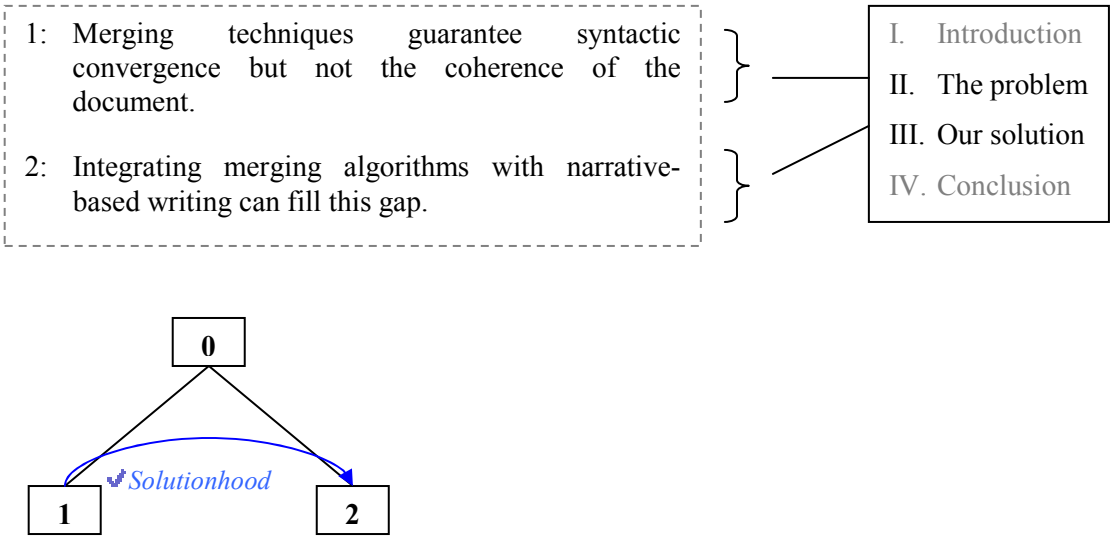


Figure 7-1: Version 1 of the DN and RS-tree (created by Author A)

The sections that need to be in the document according to the DN are listed alongside the figure. Note that the 'Introduction' and 'Conclusion' sections are mandatory for most papers and are not governed by the DN in this case (hence, they are in grey). Sections II and III planned for the document correspond to the two segments in this DN and implement the SOLUTIONHOOD relationship between them.

In theory, a paper with this structure will be sufficient. However, it is flat and lacking in detail. The general norm is to introduce some background material before talking about the problem. However, what should the material be and where should it be placed (seeing as several areas of research need to be introduced)? In our opinion, this is where a DN can play a major role. Trying to say the story, naturally, will help resolve some of these issues.

Author B responds by e-mail:

> "It's likely that many people at this conference will be from a collaborative writing background. While being aware of merging techniques, they may not know what narrative-based writing is. We should definitely include some background material on merging techniques, collaborative writing and, in particular, narrative-based writing. What do you think?"

Author B makes multiple changes to the RS-tree. She adds two new nodes and RST relationships, and creates two spans. In the tool, this would have to be done in several stages because the tool tracks and records every change in a new version. We omit these stages for the purposes of this example and label the version created by Author B as version 2.

VERSION 2 (derived from version 1 by Author B)

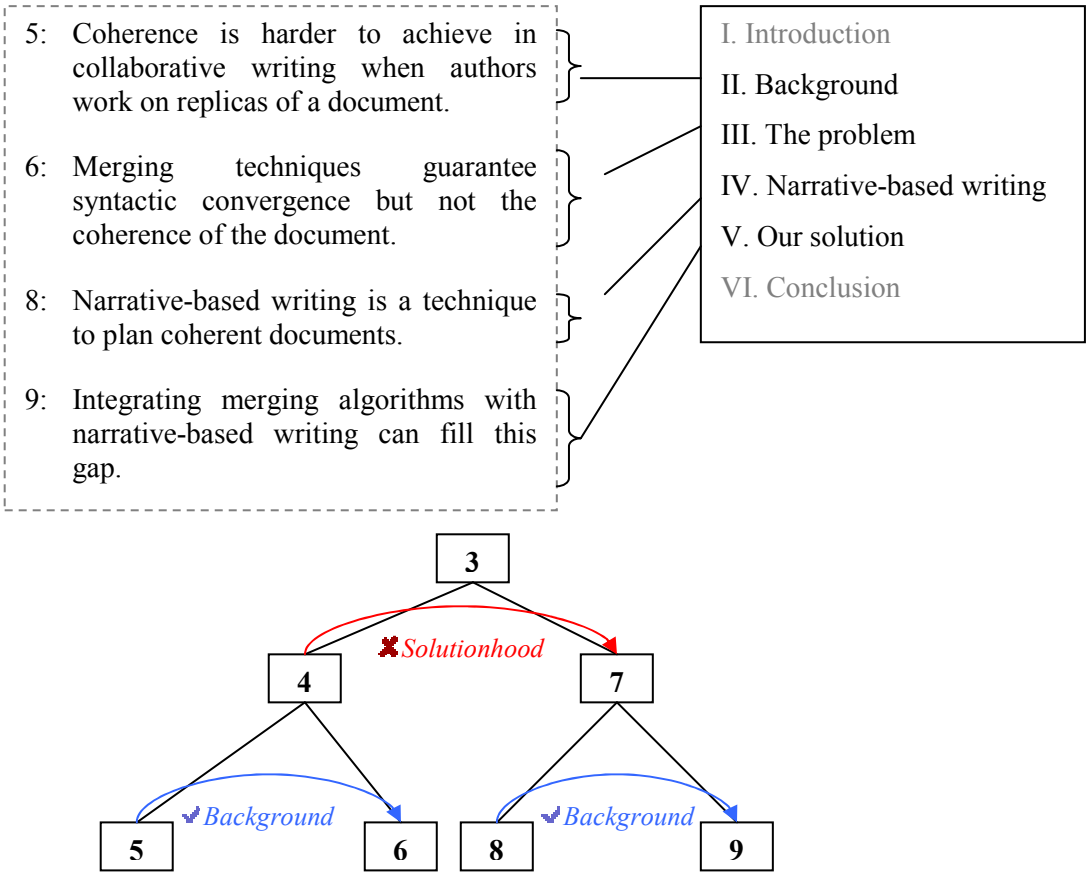


Figure 7-2: Version 2 of the DN and RS-tree (created by Author B)

Note that Author B has linked two pieces of background information into the DN. The segment about collaborative writing is the background to the problem and the segment about narrative-based writing is the background to the solution. These changes are accepted by the two other authors.

The SOLUTIONHOOD relationship is marked by the tool as being unsatisfied due to the changes made to the DN. Despite not doing a formal review of the relationships to change its state to "satisfied", the authors agree that it is still valid and get started with the writing. Authors A and B agree to do sections I, II, III and VI. Author C gets assigned sections IV and V. They are aware of how these sections should be linked (dictated by the RST relationships).

Meanwhile, Author C recognises the lack of a MOTIVATION or JUSTIFY relationship in the DN to address the 'So what? How is this useful?' question that may arise in the reader's mind. Author C adds a new node and a MOTIVATION relationship to *version 1* of the DN.

VERSION 3 (derived from version 1 by Author C)

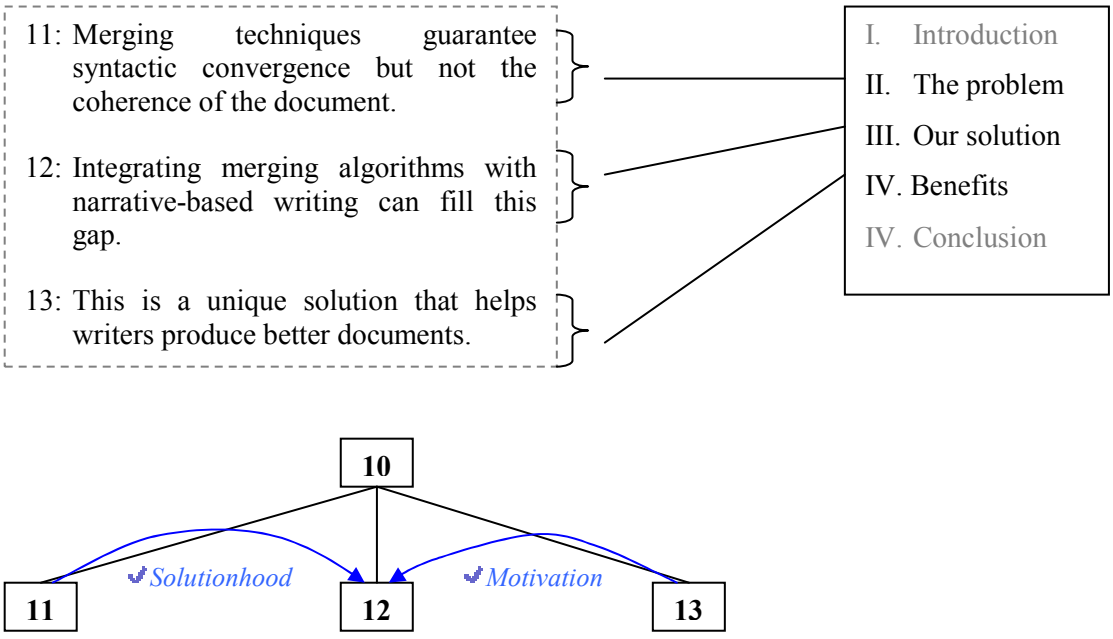


Figure 7-3: Version 3 of the DN and RS-tree (created by Author C)

The authors realise the usefulness of a MOTIVATION relationship in a DN and agree that it is an essential component of a winning paper. However, they still think the background material is important too. Seeing that version 3 was also derived from version 1 (as was version 2), they use the tool to merge the DNs to produce the results below (version 4).

VERSION 4 (merged from versions 2 and 3 by Author A)

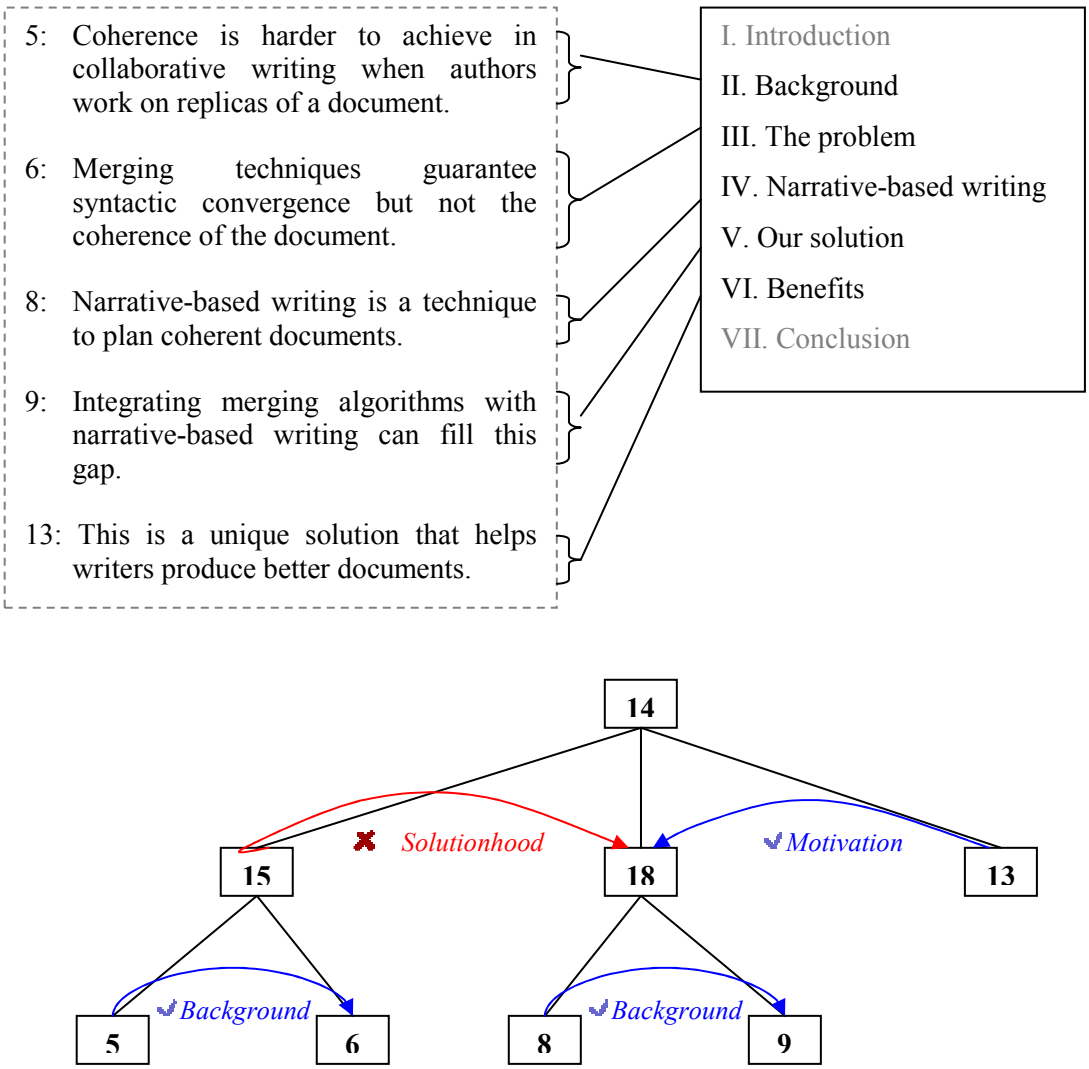


Figure 7-4: Version 4 of the DN and RS-tree (created by Author A)

The authors are happy with this merged version. The RST relationships are all still valid (even though the tool has marked SOLUTIONHOOD as "unsatisfied" according to the implemented protocol). The scene for the paper is set by the Background and Problem sections. The Background section will need to say why there is such a problem and the impact it has on documents. The solution is introduced together with a short tutorial on narrative-based writing which is necessary to fully comprehend the nature of the proposed work. The Benefits section can contain applications or examples of where the solution will help the existing situation. This will be the motivation that led the authors to develop these ideas.

For the actual paper, the DN was changed again so that the satellite of the MOTIVATION relationship preceded the solution. However, we stop the example here because the essential attributes of how the DN and the tool can assist in planning a document have been shown.

7.1.1 Discussion

Changes to the DN affect the authors' writing. In this example, the introduction of the Benefits section requires changes to the other sections of the document (which the authors had already started writing). For instance, the Problem section may now need to introduce a particular scenario which cannot be addressed by merging algorithms alone. The Benefits section can re-visit this example to show how the narrative-based approach can help the problem. This amplifies the MOTIVATION relationship.

The DN provides a way of quickly discovering the natural progression of concepts in a document. The authors need to think of the best possible story that their ideas can be fitted into. The corresponding RST analysis gives some evaluation of the story's coherence and also helps point out ill-fitting story segments or better alternatives. When several authors have opinions on the content of the paper, a DN helps combine these ideas into a coherent whole.

The tool helps manage the versions, store the RS-trees and draw the authors' attention to unsatisfied relationships. The RS-tree in the example was relatively small. A larger analysis would benefit from this tracking of the state of relationships. The tool also helps authors who are spread geographically, like in the example.

7.2 Sample applications of narrative-based writing

In this section, we present the application of narrative-based writing on a written document, a presentation and a website to emphasise that the technique can be extended to various genres of technical communication. For each, we give the DN, present the RS-tree drawn using RSTTool and show how the structure of the eventual document (or presentation or website) is influenced by the DN. The RST analysis of the DNs are not discussed at length like in the section above. However, the important points will be highlighted.

7.2.1 Research proposals

When we started our research on narrative-based writing, the research proposal was the first document we studied. Research proposals are interesting because their authors have a much harder goal to achieve: to convince the readers to fund them. Therefore, improved coherence in a research proposal is even more critical. Books on technical writing usually contain a chapter on how to write good research proposals (Alred et al., 2003, Zobel, 2004, Paradis and

Zimmerman, 2002). There are various *holistic structures*[19] (Mann et al., 1992) for them suggested by different funding bodies. For example, in 2003, EPSRC required a research proposal to contain a two-page previous research track record and a page with a diagrammatic work plan. However, the generic story required by many institutions is similar. After studying several sets of instructions on how to write a research proposal, the following generic DN was created. This DN appeared in (De-Silva and Henderson, 2005).

[We want you to fund us]1 [because we will achieve these objectives/results.]2 [We believe these results are important to you]3 [because of benefits-to-beneficiaries]4 [and to the whole world]5 [because there exists an unsolved-problem.]6 [We know this is unsolved]7 [because we have studied the background.]8 [We will solve this problem]9 [by this method.]10 [We know this is the best method]11 [because we have studied alternative-methods.]12 [To achieve this, we will need total-time]13 [and these resources]14 [because justification-of-resources.]15 [The research will be carried out by these researchers]16 [and they are the most qualified to do this because justification-of-researchers.]17 [The research will be conducted at these locations]18 [because justification-of-locations.]19

Figure 7-5: A generic DN for a research proposal that appeared in (De-Silva and Henderson, 2005)

This was our first DN. Note that it was written in the old style which included the authors' intentions and reasoning. The phrases in the DN that we expected to become sections in the document are underlined. This strong correlation between segments in the DN and section headings in the document has been abandoned. It is more important to have an *understanding* of the DN and the RST analysis prior to writing. Since the RS-tree for the DN above was too large to fit into a single figure, a collapsed version of it is given below which demonstrates the key RST relationships and also that they can be assembled into a tree structure. The four subtrees that have been collapsed are not expanded later because we move on to present a more modern version of this DN.

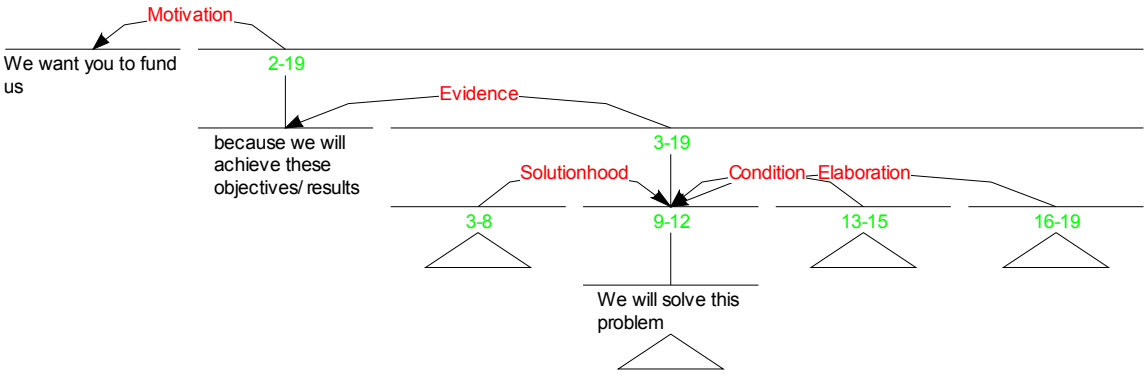


Figure 7-6: RS-tree of generic DN for a research proposal

[19] A holistic structure specifies requirements such as including an Introduction at the beginning.

A newer version of the generic research proposal DN is given below. This does not contain statements about why the authors are creating the DN in a certain way.

[We will achieve the required results in the given timeframe.][1] [These results are beneficial to you and the scientific community at large][2] [because there exists this unsolved problem to which our results are the answer.][3] [Studies into previous work in this area show that existing solutions do not address all the complexities of this problem.][4] [Our solution is unique and different to previous attempts.][5] [To achieve this, we will need total-time][6] [and these resources][7] [The research will be carried out by researchers in the following institutions][8] [because they have an impressive track record of work in this area.][9]

Figure 7-7: A new generic DN for a research proposal

The DN above has been made to reflect some ideas from the inverted-pyramid structure which was discussed in section 2.4.3. The most important part of the story (i.e. the results that will be delivered) is given first. Other details such as background research and the required resources are presented after this. An alternative would have been to introduce the unsolved problem, outline the background material and present the results. However, to achieve the goals of a research proposal, the former approach was considered better.

A possible RST analysis for this DN is given below. Once again, some subtrees have been collapsed. The figure below shows the key statement in the DN with the three other main parts of the analysis: the segments that motivate the researchers to look for the results (2-3), the segments that contain background information (4-5) and the segments that present conditions[20] upon which this research depends on (6-9).

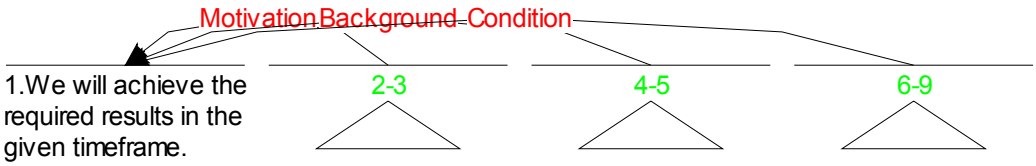
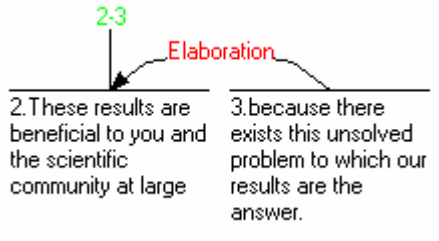


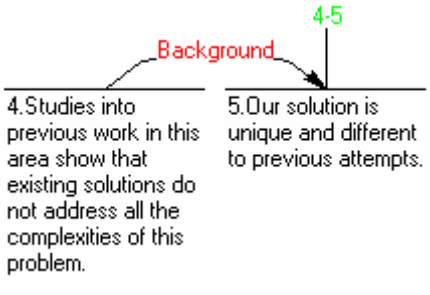
Figure 7-8: Possible RST analysis of new DN for a research proposal

[20] In this case, the CONDITION relationship (which is not in our subset of relationships for technical documents) can also be replaced with an ELABORATION relationship.

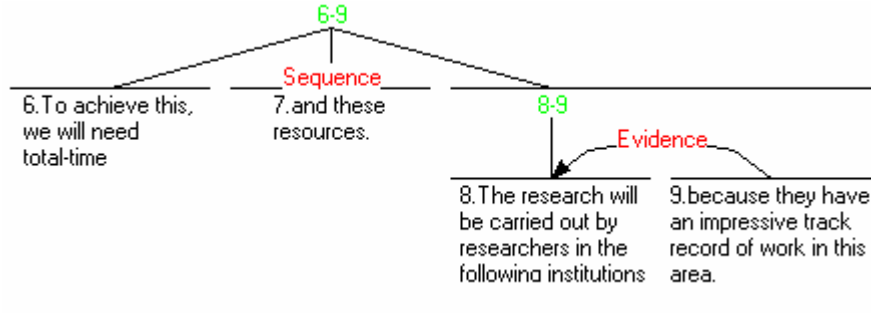
The three subtrees that were collapsed in the figure above have been expanded below.



Subtree with segments 2-3



Subtree with segments 4-5



Subtree with segments 6-9

This DN and RS-tree for a research proposal has been entered into the database of our tool. A screen shot of the tool showing this RS-tree (with all the relationships set to satisfied) is given in Figure 7-9. This DN, along with some others for popular types of documents, is available for authors to use and modify. Of course, this generic DN will need to be made more specific for an actual research proposal.

Note that the node numbers in the screenshot below do not correspond to the numbers of the segments in the RS-tree above. This is because the tool has assigned unique ID numbers to each new node. The node numbers allocated by the tool are used again in Table 7-1 which shows the corresponding sections in a research proposal.

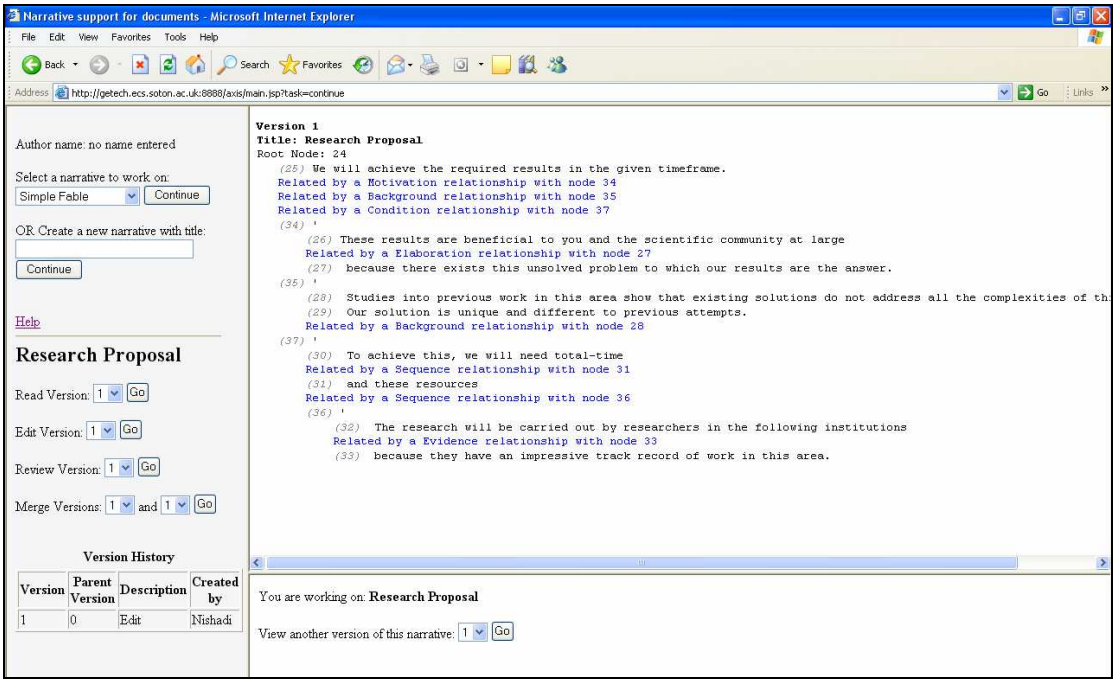


Figure 7-9: Screen shot of tool showing the RS-tree for the generic DN for a research proposal

Finally, we list below the sections in a research proposal that would correspond to the segments in the DN. (We use the node numbers from the screenshot above for the segments.)

(25) We will achieve the required results in the given timeframe.	(Introduction)
(26) These results are beneficial to you and the scientific community at large	Benefits of these results
(27) because there exists this unsolved problem to which our results are the answer.	Description of problem
(28) Studies into previous work in this area show that existing solutions do not address all the complexities of this problem.	Background research
(29) Our solution is unique and different to previous attempts.	Details of our solution (compare to existing research)
(30) To achieve this, we will need total-time	Time plan
(31) and these resources	List of resources (e.g. money)
(32) The research will be carried out by researchers in the following institutions	List of researchers
(33) because they have an impressive track record of work in this area.	Details of researchers (maybe CVs etc)

Table 7-1: Sections in a research proposal corresponding to the segments of the DN

7.2.2 Conference presentation

Another genre of scientific communication is presentations. This includes conference presentations, seminars and lectures. There are many guidelines on making a good presentation. Designing the slides clearly and pitching the content at a level suitable for the audience are some examples. While these are important issues in a presentation, they are not the topic of discussion here. We focus, instead, on the story conveyed to the audience and apply narrative-based writing to improve it.

As an example, we present a generic DN for a scientific conference presentation. This DN appeared in (Henderson and De-Silva, 2006).

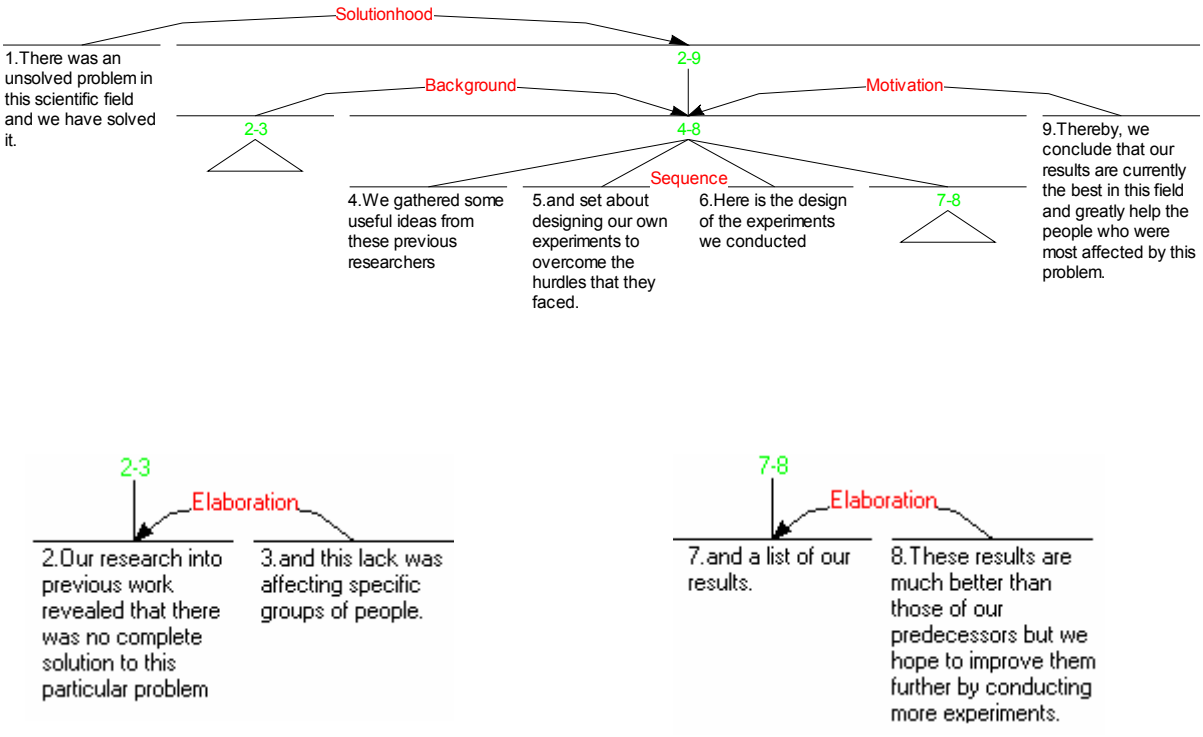
[There was an unsolved problem in this scientific field and we have solved it.]1 [Our research into previous work revealed that there was no complete solution to this particular problem]2 [and this lack was affecting specific groups of people.]3 [We gathered some useful ideas from these previous researchers]4 [and set about designing our own experiments to overcome the hurdles that they faced.]5 [Here is the design of the experiments we conducted]6 [and a list of our results.]7 [These results are much better than those of our predecessors but we hope to improve them further by conducting more experiments.]8 [Thereby, we conclude that our results are currently the best in this field and greatly help the people who were most affected by this problem.]9

Figure 7-10: Generic DN for a conference presentation

The content of this DN is similar to that of the research proposal earlier. The main difference is that in Figure 7-10 the information is presented in a more traditional fashion: problem first, then the solution and so on. In the research proposal DN, we used an inverted-pyramid like approach where the most important piece of information (in this case, the solution/results) is presented first.

The DN was divided into nine segments as shown above. A possible RST analysis is presented below. The story is divided into a problem (segment 1) and its solution (segments 2-9). This is indicated by the SOLUTIONHOOD relationship at the top of the RS-tree. Research into the current state of the problem (segments 2-3) provides *background* information and also shows that it is a significant problem worth solving. The steps in the research (segments 4-6) and the results (segments 7-8) are set in *sequence*. The fact that the results help the people affected by the problem is *motivation* to conduct this research.

The RS-tree is given below. The two collapsed subtrees in the figure are expanded later. Both the subtrees have ELABORATION relationships because the satellites provide extra information about the nuclei. This additional material is not essential but supports the understanding of the nuclei (and thus, the whole DN).



Subtree with segments 2-3

Subtree with segments 7-8

Figure 7-11: RST analysis of the DN for a conference presentation

As with writing a document, there are two aspects to the creation of a presentation. There is the *ordering* of the slides (assuming that slides are used) and the *content* that needs to be put in each of these slides. The order of the slides is determined by the order of the segments in the DN (see Table 7-2). The content of the slides and the associated speech needs to be designed according to the RST relationships (see Chapter 4).

[There was an unsolved problem in this scientific field and we have solved it.][1]	Introduction
[Our research into previous work revealed that there was no complete solution to this particular problem][2]	Background information
[and this lack was affecting specific groups of people.][3]	
[We gathered some useful ideas from these previous researchers][4]	
[and set about designing our own experiments to overcome the hurdles that they faced.][5]	Our experiments
[Here is the design of the experiments we conducted][6]	
[and a list of our results.][7]	Results
[These results are much better than those of our predecessors but we hope to improve them further by conducting more experiments.][8]	Comparison
[Thereby, we conclude that our results are currently the best in this field and greatly help the people who were most affected by this problem.][9]	Conclusion

Table 7-2: Possible list of slides that correspond to the DN

7.2.3 Project website

The story in a website may, perhaps, be the least obvious. It is harder to define and implement a DN in a website since users are free to choose their own narrative by following different links. This is not the case in documents where the ordering of the pages or sections enable the concepts to be laid out in sequence according to a well-structured narrative (Winograd, 1999). Furthermore, the impact of visual aspects such as colour and fonts is far greater in a website, making the role of a narrative appear significantly smaller.

However, it may be possible to guide users along a narrative by presenting the right menu options and having the appropriate text on each of the pages. Once again, there are popular standards for the menu items such as a 'Home' page at the start and a 'Contacts' page at the end. A DN can help determine what the other menu options should be, the order they should be in and if they need to be at the top level of navigation.

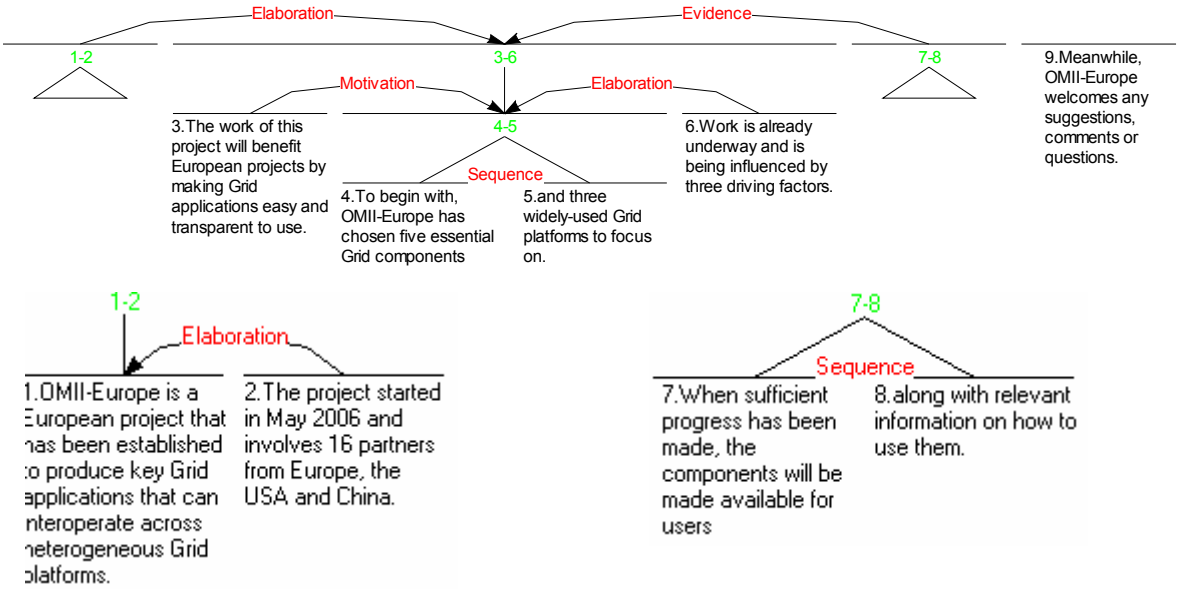
There has been some previous discussions about the narrative in a website (Bernstein, 2001, Aaronson, 2002). Of particular interest to us are the comments about users drawing conclusions about the relationships between adjacent menu items and the idea of there being a

narrative within each page as well as a narrative joining these pages together. All these are familiar concepts in narrative-based writing.

We present a DN for the website of a project called OMII-Europe. OMII-Europe stands for Open Middleware Infrastructure Institute for Europe. It is a European project funded by the EU to produce interoperable Grid components. More about the project can be found at www.omii-europe.org. The University of Southampton, UK, is the project's coordinating partner. For in-house development and discussions about the project website, we have made use of DNs. We give two versions of this DN below. The RS-trees and menu items corresponding to the segments of each DN are shown.

[OMII-Europe is a European project that has been established to produce key Grid applications that can interoperate across heterogeneous Grid platforms.][1] [The project started in May 2006 and involves 16 partners from Europe, the USA and China.][2] [The work of this project will benefit European projects by making Grid applications easy and transparent to use.][3] [To begin with, OMII-Europe has chosen five essential Grid components][4] [and three widely-used Grid platforms to focus on.][5] [Work is already underway and is being influenced by three driving factors.][6] [When sufficient progress has been made, the components will be made available for users][7] [along with relevant information on how to use them.][8] [Meanwhile, OMII-Europe welcomes any suggestions, comments or questions.][9]

Figure 7-12: DN for the OMII-Europe website (version 1)



Subtree with segments 1-2 *Subtree with segments 7-8*

Figure 7-13: Possible RST analysis for DN

The DN was divided into nine segments. In the RS-tree above, the segments are grouped into three main spans (1-2, 3-6 and 7-8). Segments 4-5 are the most important parts of the DN since they describe the work of the project. The fact that other projects will benefit from interoperable components (segment 3) is *motivation* for this work. The information about the driving factors (segment 6) *elaborates* the work. Segments 1-2, together, provide more details about the project (such as a list of the project partners) and are, therefore, involved in an ELABORATION relationship with span 3-6. Segments 7 and 8 provide *evidence* that the project is actually producing these Grid components.

Segment 9 is included in the DN because websites need to have a contacts page. It cannot, however, be fitted into the RS-tree. This is expected with certain parts of a document (e.g. the letterhead in a letter) (Mann et al., 1992) but does not mean that the DN is incoherent.

The menu items corresponding to this DN are shown below. The segments associated with each item are, respectively: 1, 2, 4, 5, 6, 7, 8 and 9. Note that there is no item relating to segment 3. According to the DN, however, there should be a third menu item called "Benefits" (or something similar) that would link to some sample applications that highlight the benefits of interoperable components. We have not yet included this because, at present, there are no specific applications to write about here since the project is still in its infancy.

Figure 7-14: A list of possible menu items (version 1)

The DN above was modified after some discussions. The "driving factors" in the project were seen to be goals. A placeholder for documents that were generated by the project was considered important and the possibility that users may look for a "download" button was raised. How should the menu items be reorganised to include these points? What's the new story? The second version of the DN is given next.

[OMII-Europe is a European project that is to produce a repository of Grid components that can interoperate across heterogeneous Grid platforms.][1] [The project involves 16 partners from around the world,][2] [all aspiring to achieve the three project goals.][3] [The focus is on re-engineering existing components. Therefore, OMII-Europe has identified five key Grid components][4] [that will be made to work across three major Grid platforms.][5] [Such interoperability benefits several European projects that rely on different infrastructures.][6] [Work is successfully underway.][7] [When sufficient progress has been made, the components][8] [and tutorials on how to use them will be posted online.][9] [Meanwhile, OMII-Europe welcomes any suggestions, comments or questions.][10]

Figure 7-15: DN for OMII-Europe website (version 2)

The second version of the DN incorporates the changes discussed above. We do not go into a detailed discussion of the RST analysis again but present the modified list of menu items below. As before, a "Benefits" button which would correspond to segment 6 has been left out. The rest of the segments all have associated menu options.

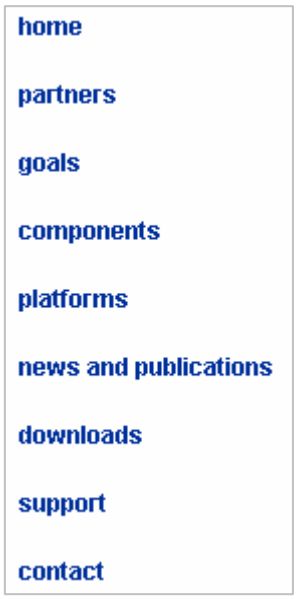


Figure 7-16: A list of possible menu items (version 2)

The preceding discussion illustrates how changes to the DN can alter the sequence of items on a menu and, thereby, influence the narrative that is imposed upon the reader (since readers are likely to assume relationships between items placed in juxtaposition). Having decided the best sequence of menu items, the second step is to create the web pages that implement the RST relationships. Some web pages may also benefit from an inverted-pyramid approach to writing where the most important information is presented in the first few lines on the page. Thus, the readers can receive the gist of the information quickly and are not forced to wade through non-essential information.

7.3 Summary

In this chapter, we presented four case studies which showed narrative-based writing being applied to various forms of technical communication.

Firstly, a collaborative writing task involving DNs and the tool was discussed. The example showed how the DN evolves due to opinions that each of the authors have about the document. When the DN changed, the sections in the document changed too. The advantages of a DN and RST analysis in this case were the converging of ideas into a coherent whole and the increased awareness among the authors about how the sections should be linked together. In chapter 2 (section 2.3.1), we discussed how co-authors could work in parallel or sequential ways. The example here demonstrated the use of narrative-based writing in a scenario where the authors worked in parallel. It could just as well support a team that worked in sequence.

The tool provided a medium by which authors could edit, analyse and merge DNs. It drew the authors' attention to unsatisfied RST relationships and managed the versions that were produced. Authors were able to revert to and compare previous versions of the DN.

Secondly, three case studies presented applications of narrative-based writing on a research proposal, a presentation and a project website. Thinking of the story in a document or presentation is natural, but it is not as obvious in a website. We do not claim that any of the DNs presented are the *best* for that genre of writing. Instead, the focus is to show how the technique and tool can be used. The creation of a document (or presentation or website) can be broadly divided into two tasks: working out the sequence of the sections and then crafting the text to fit each section. These tasks correspond, respectively, to the sequence of segments in the DN and the RST relationships.

The goal of this chapter was to show how narrative-based writing and the tool can be used. The next chapter contains an evaluation of this technique and tool.

Narrative-based writing has particular benefits in collaborative writing. The technique is not restricted to just written documents. It can be applied to presentations and websites as well.

Figure 7-17: DN for this chapter

Chapter 8

Evaluation

Chapter 7 presented four case studies that demonstrated the use of the narrative-based technique and tool in collaborative writing and different genres of technical communication.

The current chapter contains an evaluation of the technique and tool. We do this in three steps. We first describe an experiment that was conducted in May 2006 to get feedback from a group of technical authors. Next we examine the associations, if any, of narrative-based writing to technologies such as the semantic web, ontologies and speech acts. Then we compare our work to other collaborative writing and document planning approaches, re-examining our initial goals.

8.1 Experiment

An all-day experiment was conducted on the 11th of May, 2006 with nine volunteers (postgraduates and academic staff) from the School of Electronics and Computer Science at the University of Southampton, UK. The volunteers had varying amounts of experience in producing technical documents, both single-author and collaborative. The objectives, design and results of the experiment are discussed below.

8.1.1 Aims and objectives

Our aim was to get feedback on the process of narrative-based writing and the tool from technical authors. Our three primary objectives were:

- To find out if formulating a DN and doing the RST analysis helped technical authors. (How easy or difficult the RST analysis was? Does the DN help plan the structure of a document?)
- To evaluate the tool: its interface and, more importantly, the functionality offered to authors.
- To study how collaborative writing teams developed a DN and if it assisted in clarifying the ideas among the authors.

In addition to the above, we were also keen to learn if the subset of RST relationships that we identified for technical documents (see Chapter 4) was sufficient for the analyses.

8.1.2 Experiment design

The experiment began at 9:30am and carried on till 3:30pm. The five main activities of the day are outlined below.

(I) Tutorial on narrative-based writing

We presented a tutorial at the start of the experiment that described the steps in narrative-based writing (with a detailed explanation of RST), gave two examples and outlined the rest of the day's activities.

(II) RST analysis of a given DN

The volunteers were then asked to do a RST analysis of a DN for a travel brochure. The DN was provided (see Figure 8-1) so that the participants could focus entirely on the RST analysis (and not on creating the DN). This also gave rise to different analyses of the same DN which was beneficial in understanding how other technical authors perceived a DN and RST. The volunteers were, however, allowed to make minor changes to the DN if they saw it as an improvement that made the segments better fit the RS-tree (thereby, enhancing its coherence).

We want to convince the reader to book a holiday in the country described. Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for. The next page will begin with a greeting in the local language and its translation. Five to six short paragraphs will follow this, each describing attractions that will appeal to a wide range of holiday-makers; some of these attractions will be familiar and some unique so as to distinguish this country from the rest. The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations. Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader (who has read this far). Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers who may now be considering booking their holidays.

Figure 8-1: DN that the volunteers had to analyse

A travel brochure was chosen because it was a short and informal example. Note that the DN was still in the old style and contained phrases such as "the next page" and "the first of these paragraphs." It was feedback from this experiment that made us recognise that this was not ideal and change the format of DNs.

The volunteers were asked to do the analysis using the subset of RST relationships that was identified in Chapter 4. Even though the DN provided was not of a technical document, we did not anticipate that its analysis will require any additional relationships than that of a typical DN for a technical document. This enabled us to evaluate if this list was sufficient or whether the volunteers needed other relationships to complete their analysis.

(III) Enter the RST analysis from above into the tool

Each volunteer was asked to enter the analysis from the previous task into our tool. Since the RST analysis was already available, the users were free to focus entirely on the tool. The volunteers had brought their own laptops and accessed the tool via a Web browser.

(IV) Produce a DN in a team

For this task, the volunteers were divided into three teams: A, B and C. Each team was asked to produce a DN for a research paper. No other specifications were given.

(V) Fill in a questionnaire

The volunteers then had to fill in a questionnaire about the tasks above. The responses and the conclusions drawn from them are discussed next.

8.1.3 Results and conclusions

The questionnaire was divided into four sections, each focusing on a specific aspect of the feedback we wanted. See Appendix C (section C.1) for a copy of the questionnaire used.

Section 1: Information about the volunteer
Section 2: RST analysis
Section 3: Experiences using the tool
Section 4: Collaborative writing activity

The answers to each of these sections are summarised and analysed below.

Section 1

This section asked the volunteers about their writing experience, in particular if they wrote collaboratively and what methods of document planning they used. The answers are summarised in Table 8-1.

Prof - Professor
RS - Research Staff
Stu - PhD Student
O - Outlines
M - Mind maps
Doc - Document

Volunteer / Question	1	2	3	4	5	6	7	8	9
Position	RS	Stu	Prof	Stu	Stu	Stu	Stu	Stu	Stu
Docs in a month	> 5	1-5	>5	0	1-5	1-5	1-5	> 5	1-5
Of these, num of collaborative docs	Few	0	75%	0	0	0	2-3	0	0
Current doc planning technique	O	O&M	O	O	O	O	M	O	O

Table 8-1: Summary of results from the first section of the questionnaire

As seen by the answers, all the volunteers (with the exception of volunteer 4) produced documents on a regular basis. Most used outlining to plan these documents. A few volunteers regularly engaged in collaborative writing, making them ideal candidates to comment on the collaborative aspects of the narrative-based tool.

Section 2

The second part of the questionnaire was about the RST analysis of the given DN. The volunteers were asked the following questions:

- How was the tutorial at the start of the experiment?
- Did the DN dictate an appropriate structure for the travel brochure?
- How easy/difficult was the RST analysis?
- How long did it take to complete the RST analysis?
- Did you require more relationships than the ones suggested in the list?
- Were you able to form a RS-tree for the DN?
- Did you change any part of the DN to fit this RS-tree?

The responses to these questions are summarised in Table 8-2. A blank cell indicates the absence of an answer. ‘Mod’ and ‘m’ stand for ‘moderate’ and ‘minutes’, respectively.

Volunteer / Question	1	2	3	4	5	6	7	8	9
Tutorial at the start was	Good	Good	Good	Good	Good	Good	OK[1]	OK[2]	Good
Travel brochure DN appropriate?	N	Y	Y	Y	Y	Y	Y	Y	Y
Doing the RST analysis was	Hard	Mod	Hard	Mod	Hard	Mod	Mod	Easy	Mod
Time taken for analysis	45m	30m	20m	20m	25m	20m	20m	15m	15m
Required more relationships?	N	N	N	N	N	N	N	N	N
Did you form a RS-tree?	Y	Y		Y[3]	Y	Y	Y	Y	Y
Did you change the DN to fit tree?	N[4]	N		N[5]	N	Y	N	Y	N

[1] "include more examples of RST relationships"

[2] "explain how to separate a document into basic elements"

[3] "but I didn't think that the narrative was particularly easy to read"

[4] "but perhaps I would have liked to. I thought it was not allowed."

[5] "perhaps with more experience I may have done"

Table 8-2: Results from section two of the questionnaire

Feedback about the presentation at the start of the experiment was positive. Two volunteers had suggested including more examples of RST relationships and a better explanation of the segmentation process. Both comments have been taken on board for future tutorials.

The volunteers produced very different RST analyses. The RS-trees constructed by the volunteers are reproduced in Appendix C (C.2). The most common error in the RS-trees (found in about three of the analyses) was the use of relationships as shown below.

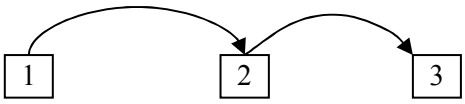


Figure 8-2: Uncommon application of RST relationships

Since this was not one of the schemas designed by Mann and Thompson (see section 3.3.1), we anticipate that such as application of relationships will not be valid in RST. A possible alternative is shown below.

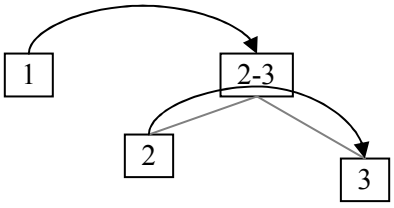


Figure 8-3: Alternative application of RST relationships

However, these technicalities of RST were not the focus of the experiment. It is likely that more details in the tutorial would have resolved this matter. The important point is that all the volunteers had managed to form RS-trees, using a range of relationships that were applicable to the given DN (SEQUENCE, MOTIVATION, ELABORATION, CONTRAST, ENABLEMENT, SOLUTIONHOOD, JUSTIFY and BACKGROUND).

With regards to doing the analysis, three volunteers found it hard. However, a majority of the volunteers had found the task moderate (i.e. not hard nor easy). After just a short tutorial teaching RST, this is actually promising. The results suggest that technical authors can be taught narrative-based writing even in a short space of time. There appears to be no apparent correlation between the experience of the writer and the ease with which he performed the RST analysis. For instance, volunteers 1 and 3 who were the most prolific technical authors found the RST analysis difficult.

None of the volunteers had said that they needed more relationships for the RST analysis. However, during the discussions after the experiment, one volunteer suggested the possibility of having an IF-THEN-ELSE relationship which he thought was useful for documents written by computer scientists. In our opinion, however, the CONDITION and OTHERWISE relationships defined in RST fulfill this need. They were not included in the list of relationships provided to the volunteers since they had not been used frequently in our previous analyses. We will consider including them in the list of relationships for technical documents.

Most volunteers thought that the DN was appropriate for the travel brochure (or at least that it resembled the DNs that we presented in the tutorial). Volunteer 1 thought that the DN was not suitable and had made this remark in the questionnaire:

> "We want *two* things – sell holiday and enable booking. This is hidden in the narrative."

Volunteer 4 had said that the DN did not read well. Subsequent discussions with the volunteers revealed that the use of phrases such as 'on the first page', 'the next page' and so on made the DN seem incoherent. It was at this point that we decided to remove such contextual information from a DN altogether. A DN is now a précis of the *story* in a document and this is the definition used in this thesis.

Section 3

This section asked the volunteers for feedback on their experience of entering the RST analysis into the tool. The volunteers had to select the functions that they used (e.g. add a node, read a version) and comment on the functionality and user interface. The following suggestions were made:

1. If the interface was graphical, it would be nice to be able to drag and drop nodes in the appropriate places in the tree.

2. Two volunteers thought it would be better if the node numbers in the tool corresponded to the node numbers assigned by the analyst (i.e. start from 1 in each RS-tree).
3. A volunteer had suggested including the capability to split existing nodes (i.e. breaking a segment into multiple smaller segments). However, this is not common in RST. Segments cannot be subdivided into smaller segments halfway through the analysis. He/she had also said: "Take a look at Eclipse based UML tools such as Rational to get some ideas on possible graphical interfaces."
4. A volunteer had proposed tagging the changes to the RS-tree with the author's name. In our tool, every change made to the RS-tree is saved in a new version and each version contains the name of the author who created that version. So, in effect, every change *is* tagged with the author's name. However, this is not common with other versioning software such as CVS which would have many changes in one version. Since this volunteer had not realised this in our tool, we may need to highlight this more in the tool's documentation or the tutorial.
5. A volunteer had also said that some user-friendliness issues may need to be addressed.

We had asked the volunteers for comments on the use of a graphical interface in the tool because, eventually, it can be an improvement to the application. However, for the scope of our research, a graphical interface does not have much added value. The comments will be saved for future work.

Section 4

The final section in the questionnaire asked the volunteers about their experience producing a DN collaboratively. The volunteers were divided into three teams: Team A (1, 2, 4), Team B (3, 5, 6) and Team C (7, 8, 9). The figures below show the three DNs that were produced for a research paper.

Team A

There is an area of scientific work that we wish to survey and bring together. There is an absence of such a survey and, as far as the foremost researchers in the field, we are the most qualified. Précis history of that area as background. We will look at the web, printed material and contact active practitioners in the field. We then correlate, categorise, structure the material and identify visible trends, gaps, conflicts, corroboration and reinforce agreements. We predict future trends in the field and identifying areas we think need further research. In the study, we have identified a significant gap in the knowledge, a conflict between two research trends and a common agreement between major research.

Figure 8-4: DN produced by team A

Team B

> We have solved an important problem. Our solution will help people in the future. There are existing solutions or partial solutions to this problem – highlight some of these solutions. Our solution is better than their's. Here is evidence of our claim based on experiments. Here is a comparison of our results with others. Here is a summary of results and claims.

Team C

> We are students of Mechanical Engineering and learning some aspects of dynamics. We wanted to verify if Newton's Law is valid with varying air friction. Originally, in Newton's law, the effect of air friction was not considered. Due to recent advances in aero dynamics, air friction measurement and its impact is a major issue. We conducted experiments X, Y and Z. As a result, we found that air friction is an important parameter affecting Newton's law. During the analysis, we found that there is a significant difference between the end velocity calculated using the formula and that produced in the experiments.

Figure 8-5: The DNs produced by the teams B and C

The three DNs produced were exceptionally good. Since the tutorial at the start contained a DN for a research proposal (see Chapter 7), we expected the DNs to be almost identical to that research proposal DN. Two of the DNs (by teams A and B) bore some resemblance and appeared to be for a generic research paper. The third DN, however, was for a specific research paper about proving Newton's law and was very different to the research proposal DN. Many of the volunteers had said that they analysed the DN using RST.

	Team A			**Team B**			**Team C**		
Volunteer / **Question**	**1**	**2**	**4**	**3**	**5**	**6**	**7**	**8**	**9**
Did writing a DN help the team?	Y	Y	Y	Y	Y	Y	Y	Y	Y
Did you analyse the DN?	Y	N	N	Y	Y	Y	Y	Y	Y
Would you use DNs in the future?	Y	Y	Y	Y	Y	Y[1]	Y	Y	Y

[1] "maybe"

Table 8-3: Results from section four of the questionnaire

Each team had taken about 20 minutes to produce the DN. Most volunteers had said that every member contributed sections of the DN and that creating a DN helped the team.

8.1.4 Summary

The experiment would have benefited from more time but it was not possible to get volunteers for a longer period than one day. However, even in this short time, the volunteers welcomed the idea of a DN for a technical document and grasped the process of doing a RST analysis surprisingly fast.

Suggestions to improve the tutorial on narrative-based writing have been noted. The volunteers did not need additional relationships to complete their analysis but we will consider adding the CONDITION relationship to the identified subset for technical documents. Comments about the user interface have also been saved but will be a part of future work because improvements to the interface (e.g. converting it to a graphical one) at this stage will not add much more to the focus of our research which is document coherence.

A significant change that took place as a result of this experiment is the difference to the style of writing a DN. We used to include information about the physical layout and authors' reasoning. After the remarks made by the volunteers, we realised that this type of information was inappropriate. Current DNs only contain a précis of the story in the document.

In conclusion, the results of this initial investigation were definitely encouraging and we have met the objectives in section 8.1.1.

8.2 Critical appraisal

This section examines the connections of narrative-based writing to related technologies. It is hardly possible to explore them all, so we have selected three technologies for this discussion: the semantic web, ontologies and speech acts.

8.2.1 Semantic Web

The WWW is a collection of documents typically written in HTML. However, HTML is incapable of adding any meaning to the content of these documents apart from basic information about the hierarchical organisation of the document (e.g. Heading 1, Body) and its presentation (e.g. font colour and size). While a human being can scan through the information to find what he is looking for (say, a list of all the good primary schools in the area), a computer or software agent is not able to do the same.

The Semantic Web is an initiative that aims to add meaning (or semantics) to these documents so that the information in them can also be processed by machines (Berners-Lee et al., 2001). In order to do this, technologies such as XML, Web Ontology Language (OWL) and Resource Description Framework (RDF) are being used.

XML allows everyone to create their own tags; thus increasing flexibility. Ontologies allow information in different databases to be shared unambiguously. RDF is based upon making statements about knowledge (or resources) in the form of subject-predicate-object triples. The subject and object usually denote resources (identified by Uniform Resource Identifiers (URI)) while the predicate expresses a relationship between them (e.g Banister Infant School, located-in, Southampton).

RDF triples are similar, in our opinion, to RST relationships. We envisage that, just as resources are linked using relationships such as "is-a-friend-of", they can be linked via RST relationships such as MOTIVATION and BACKGROUND too. One advantage of RST relationships is that they have fixed definitions and, therefore, will mean the same thing across databases. So, the subject and object of a RDF triple would be the nucleus and satellite of a RST relationship.

As can be seen, there are definite parallels that can be drawn between the two areas of research and both communities can learn from each other. Perhaps the use of RST can enable software agents to automatically evaluate the level of coherence of documents (or information on the whole) by navigating the RST links and looking for RS-tree structures. Users can also be presented with information like: "Here's the background information to that particular problem and the motivation behind solving it." We do not pursue these areas of research here but they are interesting possibilities for future work.

8.2.2 Ontologies

In philosophy, the word "ontology" is the study of being or existence. Artificial Intelligence (AI) and Web researchers use the word to refer to a document or file that formally defines the relationships between terms (Berners-Lee et al., 2001). Ontologies define objects, the classes they belong to, their attributes and relationships to other objects. For instance, going back to the primary school example from the previous section, the attributes of a primary school can include its name, address and the number of pupils. Each primary school is a subclass of "school" which can include secondary schools and so on. Such definitions form a taxonomy.

A taxonomy can be complemented with inference rules. For example, a simple rule that says "If the number of pupils in the school is more than a hundred, it is a big school" can help software automatically list only the significant primary schools in the area. Other information that is likely to be misinterpreted across databases (such as a postcode in the UK and a zip code in America) can also be resolved using ontologies.

The use of ontologies in narratives is not uncommon. For instance, Tuffield et al (2006) discuss an ontological understanding of narratives and Bärenfänger et al (2006) talk about a taxonomy of RST relationships that will help discourse parsing. We too started out thinking that document structures could be defined using an ontology based on the idea that sections in a document, while having attributes of their own, were also linked to each other. We even went on to implement a simple ontology editor that allowed users to create (and reuse)

document structures. Reverting to the story of the Fido and the Flea in Chapter 2 (section 2.6.1), the ontology had a set of triples such as the ones shown below to model the story events and characters.

<Fido>, <is-a>, <Dog>
<Fido>, <gets>, <Fleas>
<Dog>, <is-a>, <Animal>

Ontologies can also be made to contain information such as: "A research proposal should contain an Introduction, a Background section…and the Introduction should be linked to the Background in the following way." However, we eventually moved away from ontologies as the focus of our research shifted towards document coherence. Nevertheless, ontologies are a possible way of modelling RS-tree structures as well and can be an area for future work. Ontologies can enable agents to recognise certain types of documents (E.g. If a document has the following sections and an executive summary, it must be a research proposal). Moreover, with the use of RST, it may be possible to get them to recognise whether or not that research proposal is coherent.

8.2.3 Speech acts

A "speech act" is a term from linguistics and the philosophy of language. It is based on the idea that in saying something, we *do* something. More formally, speech acts "designate all intentional actions…carried out in the course of a communication" (Ferber, 1999). Some examples include: it is raining, wash your hands and I promise I'll be back by five.

John Searle is a prominent figure in this area. He introduced, in particular, the idea of indirect speech acts. An indirect speech is an utterance such as "Would you mind turning down the stereo?" which appears to warrant a Yes/No answer but will usually result in the hearer turning down the volume instead. Searle developed a series of steps that explained how two meanings can be derived from the same utterance.

Speech acts have been influential in AI for communication between software agents. More importantly for us, speech acts have been linked to technical writing too. James Euler (1992) states that a technical document is a conversation between the writer (or writers in our case) and the reader through which they achieve some act (e.g. use some software, get funding). He goes on to argue that, in this respect, "voiceless" technical writing is unfair. Technical documents are required to be "depersonalised" (without the use of 'I' or 'We') when, in reality, many of them have personal goals (such as to please the writer's employers).

Euler's discussion is not too far from our work. The purpose of our research has been to improve this conversation between reader and writer (by enhancing the coherence). Our early DNs included explicit statements such as "We want you to fund us" which are, in a sense, speech acts. Most of these statements have now been removed from a DN but many of the intentions (and assertions) are still encapsulated in the RST relationships (e.g. this motivates

the reader to do something). There are many aspects of speech acts (and the corresponding theories) that can have some bearing on narrative-based writing. However, we do not have time to explore this here but list it, once again, as a possible area for future research.

Depending on one's experience and research background, narrative-based writing can be related to several other technologies. We have chosen three that seemed the most relevant but do not claim that this is an exhaustive list. All these comparisons expose many interesting areas of work which we do not have time to go in to now, but list as future research.

8.3 Our initial goals revisited

For the final part of our evaluation, we re-examine the research goals we set for ourselves in Chapter 2. We recognised that document coherence was linked to the story conveyed to the readers and that this was difficult to get right in collaborative writing. We realised that current planning techniques and writing tools did not address the issue of document coherence, and set out to develop a new narrative-based planning technique and tool.

In Chapter 2, we investigated three planning techniques: mind maps, outlines and the pyramid principle. Mind maps provided a good visual aid to authors but had to eventually be converted to a linear format in order for it to be useful in the actual writing of the document. Mind maps were also subjective. So, a map drawn by one author could be misunderstood by another author. Outlining was the most popular technique. Outlines provided a way in which the sections of a document could be laid out in sequence, but there were no explicit relationships identified between these sections (apart from that they were in sequence and that some sections were contained within others). The pyramid principle was more elaborate than either of the previous techniques. It had definite instructions on how to structure the arguments in a document and construct a logical flow. It was also unique in that it encouraged authors to think of the document from a *reader's* point of view. The pyramid structures came the closest to addressing document coherence and we have used some of its properties *together* with our narrative-based approach. For instance, version two of the research proposal DN in Chapter 7 (section 7.2.1) was written in an inverted-pyramid structure. The only criticism of the pyramid principle was that it was relatively complex. All three methods also did not allow an author to judge if one structure for a document was more coherent than the other.

Each of the techniques had particular benefits that we were keen to include in any new technique that we developed. These features were:

- Provide a visual aid
- Provide a way of determining the natural, linear ordering of the sections in a document
- Connect these sections logically

In addition to these, we also wanted to look for ways in which the authors could judge if their document was coherent.

Narrative-based writing was designed to address these issues. When authors start working out the *story* in the document, they automatically formulate the best linear order for the sections in a document (since the sequence of sections in the document corresponds to the sequence of segments in the DN). The RST analysis helps connect these sections with logical relationships and the eventual RS-tree is a good visual aid. We have also suggested that DNs should be kept short so that the RS-trees are smaller and easier to manipulate. The assertion in RST of a tree structure helps authors gauge the quality of their DN (and thus, the coherence of the document).

RST dictates that if all the segments in the document can be linked via relationships (and more importantly, formed into a tree) then the text is coherent. When there are segments that cannot be included in the RS-tree, it is an indication to the authors to rethink the DN. Of course, there are some segments in a DN that are not expected to fit in the tree (such as the segment corresponding to the letterhead in a letter or the contacts page in a website). This is normal.

The main disadvantage with narrative-based writing is having to learn RST. Most technical authors are not going to be familiar with the use of a discourse theory. Initially, even reading RS-trees is not entirely straightforward (especially those that are drawn by RSTTool) since they are different to traditional tree structures in computer science. However, the results of the experiment have been encouraging. The nine technical authors who took part learnt RST very quickly. The minor irregularities in the RST applications (see Figure 8-2) could have been avoided if there had been a longer, more comprehensive tutorial at the start.

Having developed this technique, we went on to design and implement a tool. The most that current writing tools did towards enhancing coherence was provide templates for certain genres of documents (e.g. Newnovelist, the wizards in Microsoft Word) and ensure that the replicas of a document that the authors were working on individually were kept syntactically equivalent (e.g. operational transformation). It needs to be said that, apart from coherence, the other aspects of collaborative writing such as version control and merging have all been well established in these other tools (e.g. CVS).

The aim of our tool was to enable a team of geographically-dispersed authors to engage in narrative-based writing. By implementing this tool, we have had to address the non-trivial issues surrounding the manipulation of versions of RS-trees (e.g. maintaining the sequence of nodes using NXT relationships). The main contribution of our work has been the identification of a set of functions that allows the creation, analysis and reviewing of a DN. The tool was implemented as a Web-based application using JSP, HTML, Java and a relational database.

Our tool focuses on the DN and RS-tree. The third component of this process, the actual document, is not dealt with by us since several other tools such as CVS handle collaborative documents well. It is anticipated, therefore, that our narrative-driven functions can, in the future, be added on to these existing tools so that they support coherence as well (or at least

allow authors to plan coherent DNs). Some discussions in this area have already begun (De-Silva and Skaf-Molli, 2006).

We appreciate that there are user interface issues that need to be addressed in our tool. Some points about this were raised during the experiment too. Visually, a graphical representation of RS-trees may be better. Also, the rapidly increasing node numbers become confusing after a while. It will be better if each RS-tree could be displayed with node numbers corresponding to how the author divided the DN into segments. However, while these issues are important, they are not essential to the point we are trying to make. Our focus was to devise a tool that allowed authors to share DNs and RS-trees so that their ideas can be moulded into one coherent story. We have made the first steps towards achieving this goal.

8.4 Summary

This chapter contained an evaluation of narrative-based writing and our tool. We first described an experiment that was conducted in May 2006 with nine volunteers. The volunteers were assigned set tasks that focused on RST, the tool and collaborative writing. The results were definitely encouraging. The participants understood and welcomed the concept of a DN and, even with a relatively short introduction to RST, managed to complete the RST analyses. We got useful feedback about DNs and the tool. We have implemented some comments straightaway and left the rest for future work.

Secondly, we discussed three technologies just outside the scope of our research that we were able to see had connections to narrative-based writing. Several parallels can be drawn with technologies such as the semantic web. We are not able to explore all these areas in the time given but leave them in the thesis as possible future work.

Finally, we re-examined our initial research goals from Chapter 2. We conclude that narrative-based writing addresses the criteria that we identified as being essential in a planning technique and that the tool has fulfilled its objective of enabling authors to engage in narrative-based writing, albeit requiring some improvements to its interface.

In the next chapter, we outline some more areas of future work and present our conclusions.

We have evaluated our technique and tool in three steps. We conducted an experiment with some technical authors who gave us feedback on narrative-based writing and the tool. We then drew some parallels between our work and other areas of research that we believe can expose interesting future research prospects. Finally, we re-examined our research goals.

Figure 8-6: DN for this chapter

Chapter 9

Conclusions and future work

There is an increasing demand for technical documents and, often, they need to be produced collaboratively with peers against tight deadlines. The title 'technical document' in this thesis was used to refer to a variety of forms of communication in a scientific context including written documents, presentations and websites. For these documents to fully achieve their respective goals, they need to, of course, be technically sound, well presented and free of spelling and grammatical mistakes. However, perhaps the most important aspects of an effective document are consistency and coherence.

Document coherence is a subjective phenomenon. We defined coherence as the attribute of a text that makes it understandable and easy to follow. The *order* in which the sentences are placed can have a significant impact on coherence, even if each individual sentence is perfectly constructed. This was illustrated in Chapter 2. Incoherence is easy to detect in short texts such as a paragraph but this is not the case with large documents, particularly if they are produced by multiple authors. Even if the problems are recognised in such documents, it may not always be obvious how to correct them.

Authors are usually encouraged to make use of some planning techniques prior to writing to organise their ideas. Outlines, mind-maps and the pyramid principle are just three of these techniques. While each of them has individual strengths, they do not help authors work out the natural ordering of their ideas and there was, definitely, no way of checking if the sequence of sections formed a coherent text. We anticipated that current writing tools would provide co-authors with some assistance towards document coherence. However, while the tools facilitate collaborative working excellently, they lack explicit support or guidance for the semantic coherence in documents.

After reading several texts that gave advice on technical writing (e.g. (Zobel, 2004)), we picked up on the idea that a document should convey a narrative (or story) and decided that more could be done to help technical authors ensure this story was consistent. While doing research on narratives, we discovered that linguists had developed discourse theories, such as RST (Chapter 3), that helped analyse and synthesise coherent narratives. The combination of

ideas from these parallel strands of research formed the basis for our new technique called **narrative-based writing** (Chapter 4).

In this technique we introduced the concept of a **document narrative (DN)** which is an explicit précis of the story that a document conveys to its reader. A DN could be further analysed using RST to ensure that it is coherent (i.e. Do the relationships in it assemble into a tree?) and add more meaning (e.g. Information in section A is the motivation for conducting this research). The third step in the technique is to use the DN and the RST analysis to structure the document.

We built a Web-based tool to enable teams of authors to engage in narrative-based writing. As for all tools that supported collaborative working, particularly with non-trivial data structures such as RS-trees, careful design was crucial. We used a combination of three models to design our tool (Chapter 5). The main contribution made by our design is a set of functions that enable collaborative narrative-based writing. These functions were implemented in Java as a proof of concept (Chapter 6). The resulting tool enables authors to edit, analyse, review and merge DNs asynchronously via the Web.

The technique and tool have been used to produce structures for several documents. A few of these were presented in Chapter 7 as case studies. Finally, we conducted an experiment using a group of volunteers to evaluate the technique and our tool (Chapter 8). The results of the experiment were definitely encouraging. Even with just a short tutorial, the volunteers learned the concepts of the DN and the RST analysis quickly. We received useful feedback about DNs and the interface of the tool; some of which have already being implemented.

We now summarise the main contributions of our research, and present the future work ideas and concluding remarks.

9.1 List of our main contributions

The contributions have been divided into primary and secondary contributions.

- **(PRIMARY) A new technique called narrative-based writing**

  A new technique called **narrative-based writing** was introduced that enables authors to improve the coherence of collaborative technical documents. The technique uses ideas from narratives and RST and applies them to technical writing. The use of RST in this context (i.e. to *synthesise* technical documents) differs greatly to its mainly analytical applications in the past. Narrative-based writing was designed to address the shortcomings of the other document planning techniques by providing a way of working out the natural sequence of ideas in a document and evaluating the coherence of the implicit story by using RST.

- **(PRIMARY) The design for a narrative-based tool for collaborative writers**

  Chapter 5 presented the combination of three models that was used to clarify the concepts and design a tool that supports collaborative narrative-based writing. The chapter also discussed the data structures necessary to store the RS-trees. This design furthers the understanding of narrative-based writing and addresses the complexities involved in the manipulation of RS-trees.

- **(PRIMARY) A web-based tool for collaborative narrative-based writing**

  The third primary contribution made by our research is the web-based tool. It has been built and tested as proof of concept of the design. Chapter 6 discussed how the tool was implemented and the choices made between alternative technologies.

- **(SECONDARY) A tutorial and a catalogue of case studies**

  The tutorial we produced for the experiment to teach narrative-based writing is available online (www.narratives-uk.com) and is in the process of being substantially expanded. Several DNs (and corresponding RST analyses) for various types of documents have also been produced. Some of them were included as case studies in Chapter 7. The tutorial and these sample DNs can be useful guidelines for technical authors. Already, the DN for the abstract of a research paper (see Appendix A, section A.3) has assisted a few colleagues in the lab with their writing.

- **(SECONDARY) Evaluation via an experiment and critical appraisal**

  The tool and the technique have been evaluated by us and also by a group of volunteers in an experiment conducted in May 2006. The outcomes of the experiment were encouraging.

9.2 Summary of our main contributions

RST is a formal method of analysing texts. In narrative-based writing, we use the ideas from RST in the *synthesis* of technical documents. By making the authors attend to the RST analysis of the DN, they are forced to think about the structure and story in a greater level of detail. This eventually leads to improved document coherence. Coherence is an important issue in documents which has not been dealt with before in this way. By implementing the tool, we had to study and solve all the issues surrounding narrative-based writing in a collaborative scenario. The functions that arose may not be highly efficient or the most elegant, but they address the necessary aspects of a collaborative RST analysis in a complete manner. Of course, there are improvements that could be made. However, achieving all these goals within the three years of PhD research is nearly impossible. The initial objectives we set ourselves have been met and we are investigating the areas of future work.

9.3 Future work directions

The future work directions we intend to pursue are outlined below. We have already made some progress in some of these areas.

- **Enhancements to the functions**

  The enhancements to the functions are twofold. First, we will improve the functions we have got already by adding more RST-related *rules* or *guidelines*. Secondly, some new, more elaborate, functions will be investigated. Both these improvements are discussed below.

  (a) As an example of the first type of enhancement, we demonstrate the adding of a segment below. Imagine a RS-tree as shown below.

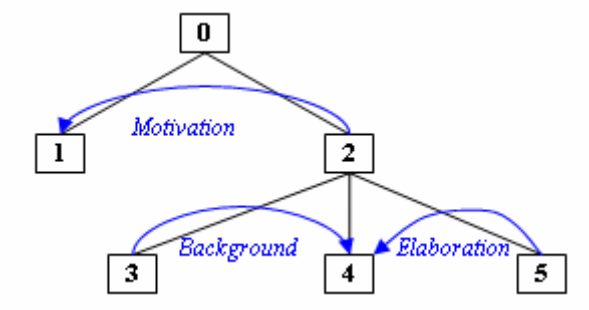


Figure 9-1: Initial RS-tree

  If a node is added before node 3 (say, 3A), the current function will allow the following tree to be formed with the BACKGROUND and MOTIVATION relationships set to 'unsatisfied' (to alert the authors to closely scrutinise and correct the relationships).

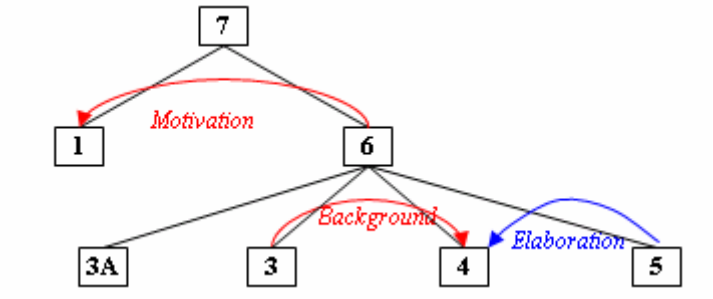


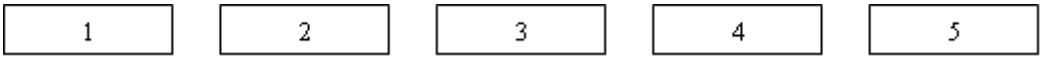
Figure 9-2: Tree after a node is inserted

  An improvement to the function will be to recognise that the BACKGROUND relationship is now no longer between the second and third segments in the DN, and inform the author of how this will affect the document. For instance, should the

BACKGROUND relationship now be between node 3A and 3? If so, 3A has to be the satellite since it has been suggested by Mann and Thompson that, for a BACKGROUND relationship, the satellite should be presented before the nucleus. The function could remove the BACKGROUND relationship or ask the author for confirmation. These rules are not essential at the moment since authors are given complete control over the maintenance of relationships.

(b) The second kind of enhancement is the inclusion of more elaborate functions to the existing suite. One function that we have begun to study is explained below. This function takes as arguments, a sequence of DN segments and the set of RST relationships between them. From these values, the function will be able to suggest a possible RS-tree.

For example, imagine a sequence of five segments in the DN.



An author recognises that the following relationships exist between these segments. The relationships are not named in order to simplify the diagram.

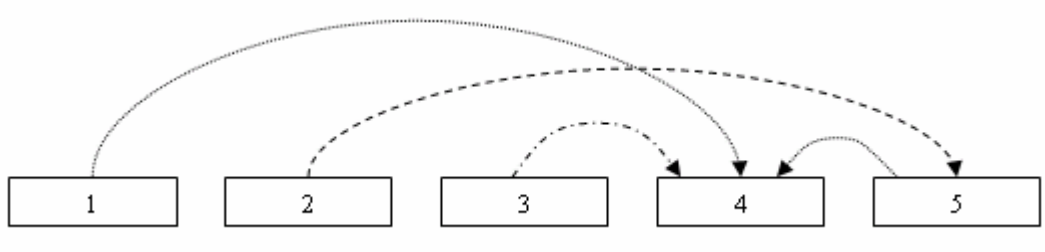


Figure 9-3: Relationships between segments of a DN

A possible RS-tree that could be generated from these relationships is shown below. In a sense, this can be an automatic verification of coherence. Defining the set of rules and assumptions that need to be made to construct such a tree is not trivial.

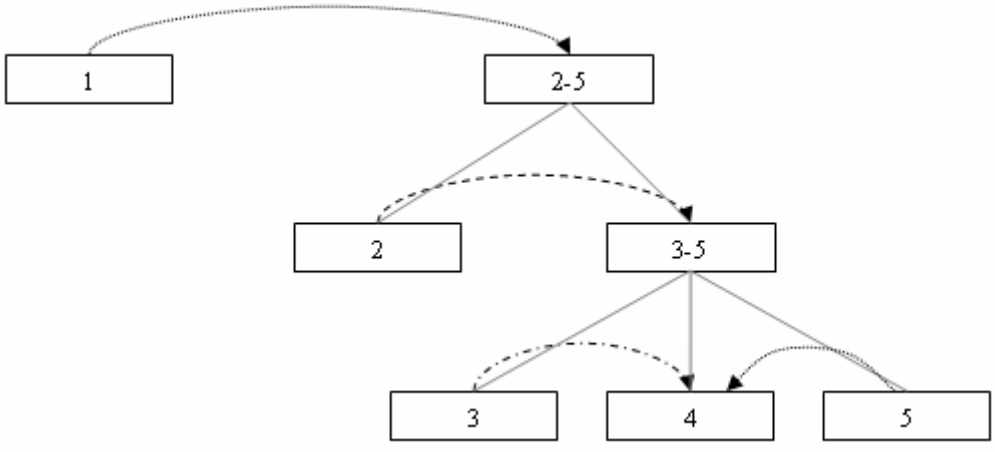


Figure 9-4: Possible RS-tree that incorporates all the relationships above

- **Integrate narrative-based writing into existing tools**

  Existing collaborative working tools such as CVS or LibreSource already possess advanced versioning and merging properties. We anticipate that if narrative-based support could be integrated into these tools, the co-authors can have documents that are both syntactically merged and following a coherent DN. We have already made some headway in this regard by collaborating with the researchers at the University of Nancy, France who are involved in the development of LibreSource. Our research ideas, still in their infancy, are in (De-Silva and Skaf-Molli, 2006) where we explore the possibilities of using the merging technique called Operational Transformation (OT) to converge copies of a DN and RS-tree that authors may be editing simultaneously.

- **Identify recurring patterns of relationships**

  Having analysed DNs for several technical documents, it may be possible to identify recurring patterns in the ways in which these relationships are commonly assembled. For instance, the Problem-Solution narrative is often used in technical documents. A set of relationships associated with this narrative is shown below. We have alluded to some aspects of this structure in the collaborative writing scenario in Chapter 7 (when Author C realises that a MOTIVATION relationship is missing).

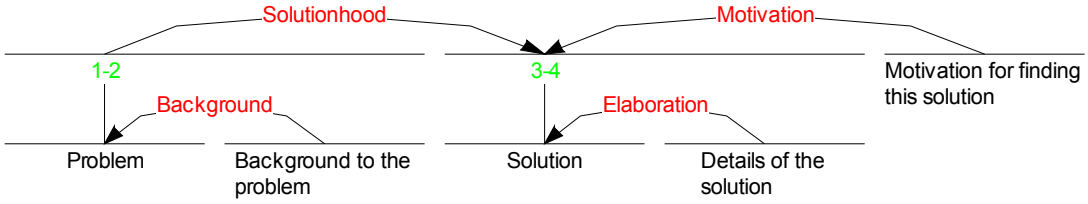


Figure 9-5: Regular Problem-solution pattern in technical documents

If more patterns can be identified, the tool can be further enhanced to give some guidance to the authors when creating and analysing DNs.

9.4 Concluding remarks

We set out to address the problem of incoherence in co-authored technical documents. We have achieved this goal by combining the ideas from narratives and RST into a new planning technique and tool for authors. After encouraging results from our own evaluation and from experimental evaluation, we are keen to put our theories into practice. We are currently exploring the impact of DNs on the website and documentation produced in OMII-Europe[21]. Software support for narrative-based writing, particularly its inclusion in popular collaborative tools such as CVS, can greatly influence the ways in which co-authors coordinate their ideas and enhance the quality of the eventual documents.

We have used narrative-based writing to structure the content of this thesis. We end by presenting, once again, the DN for the entire thesis (the framing DN). Each segment in the DN below corresponds to one of the chapters.

[We believe that a narrative-based approach can help technical authors improve the coherence of documents they produce collaboratively.]1 [Coherence can be attributed to the story conveyed by a document. It is particularly difficult to get right in collaborative technical writing. Current writing tools do not support document coherence.]2 [Narrative and discourse theories, in particular RST, provide a solution.]3 [By combining the knowledge of these two parallel strands of research (narratives and technical writing), we have developed a new method of document structuring called narrative-based writing.]4 [In order to facilitate teams of geographically-dispersed authors to engage in narrative-based writing, we have carefully designed a tool]5 [and done a Web-based implementation of it.]6 [The new technique and tool are particularly beneficial in collaborative writing and can also be applied to other genres of technical communication such as websites and presentations.]7 [Preliminary investigations suggest that the narrative-based approach is helpful]8 [and that the tool, with some enhancements, can be a valuable contribution to technical authors.]9

Figure 9-6: DN for the thesis

[21] Open Middleware Infrastructure Institute for Europe (www.omii-europe.org)

Appendix A

RST definitions and analyses

This appendix contains the definitions for all the RST relationships, the RS-trees for the DNs that have not been analysed elsewhere in the thesis and the DN for an abstract of a research paper (which was mentioned in Chapter 9).

A.1 Definitions of the RST relationships

Below we reproduce the definitions for the 23 relationships from the original RST paper (Mann and Thompson, 1988). First the hypotactic relationships (one nucleus, one satellite) are defined followed by the paratactic relationships (multiple nuclei). Each definition consists of four fields:

1. Constraints on the Nucleus
2. Constraints on the Satellite
3. Constraints on the combination of the Nucleus and Satellite
4. The Effect

(N – nucleus, S – Satellite, R – Reader, W – Writer)

A.1.1 Hypotactic relationships

EVIDENCE	
Constraints on the Nucleus:	The reader R might not believe the information that is conveyed by the nucleus N to a degree satisfactory to the writer W
Constraints on the Satellite:	The reader believes the information that is conveyed by the satellite S or will find it credible
Constraints on N+S combination:	R's comprehending S increases the R's belief of N
The effect:	R's belief of N is increased
Locus of the effect:	N

JUSTIFY	
Constraints on the Nucleus:	None
Constraints on the Satellite:	None
Constraints on N+S combination:	R's comprehending S increases the R's readiness to accept W's right to present N
The effect:	R's readiness to accept W's right to present N is increased
Locus of the effect:	N
SOLUTIONHOOD	
Constraints on the Nucleus:	None
Constraints on the Satellite:	Presents the problem
Constraints on N+S combination:	The situation presented in N is a solution to the problem stated in S
The effect:	R recognizes the situation presented in N as a solution to the problem presented in S
Locus of the effect:	N and S
ELABORATION	
Constraints on the Nucleus:	None
Constraints on the Satellite:	None
Constraints on N+S combination:	S presents additional detail about the situation or some element of subject matter which is presented in N or inferentially accessible in N in one or more of the way listed below. In the list, if N presents the first member of any pair, then S includes the second. set : member abstract : instance whole : part process : set object : attribute generalization : specific
The effect:	R recognizes the situation presented in S as providing detail for N. R identifies the element of subject matter for which detail is provided.
Locus of the effect:	N and S
BACKGROUND	
Constraints on the Nucleus:	R won't comprehend N sufficiently before reading text of S
Constraints on the Satellite:	None
Constraints on N+S combination:	S increases the ability of R to comprehend an element in N
The effect:	R's ability to comprehend N increases
Locus of the effect:	N

ENABLEMENT	
Constraints on the Nucleus:	Presents R action (including accepting an offer), unrealized with respect to the context of N
Constraints on the Satellite:	None
Constraints on N+S combination:	R comprehending S increases R's potential ability to perform the action presented in N
The effect:	R's potential ability to perform the action presented in N increases
Locus of the effect:	N
MOTIVATION	
Constraints on the Nucleus:	Presents an action in which R is the actor (including accepting an offer), unrealized with respect to the context of N
Constraints on the Satellite:	None
Constraints on N+S combination:	Comprehending S increases R's desire to perform action presented in N
The effect:	R's desire to perform action presented in N is increased
Locus of the effect:	N
CIRCUMSTANCE	
Constraints on the Nucleus:	None
Constraints on the Satellite:	S presents a situation (not unrealized)
Constraints on N+S combination:	S sets a framework in the subject matter within which R is intended to interpret the situation presented in N
The effect:	R recognises that the situation presented in S provides the framework for interpreting N
Locus of the effect:	N and S
VOLITIONAL CAUSE	
Constraints on the Nucleus:	Presents a volitional action or situation that could have arisen from a volitional action
Constraints on the Satellite:	None
Constraints on N+S combination:	S presents a situation that could have caused the agent of the volitional action in N to perform that action; without the presentation of S, R might not regard the action as motivated or know the particular motivation; N is more central to W's purposes in putting forth the N-S combination than is S
The effect:	R recognises that the situation presented in S as a cause for the volitional action presented in N
Locus of the effect:	N and S

NON-VOLITIONAL CAUSE	
Constraints on the Nucleus:	Presents a situation that is not a volitional action
Constraints on the Satellite:	None
Constraints on N+S combination:	S presents a situation that, by means other than motivating a volitional action, caused the situation presented in N; without the presentation of S, R might not know the particular cause of the situation; a presentation of N is more central than S to W's purposes in putting forth the N-S combination.
The effect:	R recognises the situation presented in S as a cause of the situation presented in N
Locus of the effect:	N and S
VOLITIONAL RESULT	
Constraints on the Nucleus:	None
Constraints on the Satellite:	Presents a volitional action or a situation that could have arisen from a volitional action
Constraints on N+S combination:	N presents a situation that could have caused the situation presented in S; the situation presented in N is more central to W's purposes than is that presented in S
The effect:	R recognises the situation presented in N could be a cause for the action or situation presented in S
Locus of the effect:	N and S
NON-VOLITIONAL RESULT	
Constraints on the Nucleus:	None
Constraints on the Satellite:	Presents a situation that is not a volitional action
Constraints on N+S combination:	N presents a situation that caused the situation presented in S; presentation of N is more central to W's purposes in putting forth the N-S combination than is the presentation of S.
The effect:	R recognises the situation presented in N could have caused the situation presented in S
Locus of the effect:	N and S
PURPOSE	
Constraints on the Nucleus:	Presents an activity
Constraints on the Satellite:	Presents a situation that is unrealised
Constraints on N+S combination:	S presents a situation to be realized through the activity in N
The effect:	R recognises that the activity in N is initiated in order to realize S
Locus of the effect:	N and S

ANTITHESIS	
Constraints on the Nucleus:	W has positive regard for the situation presented in N
Constraints on the Satellite:	None
Constraints on N+S combination:	The situations presented in N and S are in contrast (cf. *CONTRAST,* i.e. are (a) comprehended as the same in many respects, (b) comprehended as differing in a few respects and (c) compared with respect to one or more of these differences); because of an incompatibility that arises from the contrast, one cannot have positive regard for both the situations presented in N and S; comprehending S and the incompatibility between the situations presented in N and S increases R's positive regard for the situation presented in N
The effect:	R's positive regard for N is increased
Locus of the effect:	N
CONCESSION	
Constraints on the Nucleus:	W has positive regard for the situation presented in N
Constraints on the Satellite:	W is not claiming that the situation presented in S doesn't hold
Constraints on N+S combination:	W acknowledges a potential or apparent incompatibility between the situations presented in N and S; W regards the situations presented in N and S as compatible; recognizing that the compatibility between the situation presented in N and S increases R's positive regard for the situation presented in N
The effect:	R's positive regard for the situation presented in N is increased
Locus of the effect:	N and S
CONDITION	
Constraints on the Nucleus:	None
Constraints on the Satellite:	S presents a hypothetical future or otherwise unrealized situation (relative to the situational context of S)
Constraints on N+S combination:	Realization of the situation presented in N depends on realization of that presented in S
The effect:	R recognizes how the realization of the situation presented in N depends on the realization of the situation presented in S
Locus of the effect:	N and S
OTHERWISE	
Constraints on the Nucleus:	Presents an unrealized situation
Constraints on the Satellite:	Presents an unrealized situation
Constraints on N+S combination:	Realization of the situation presented in N prevents realization of the situation presented in S
The effect:	R recognizes the dependency relation of prevention between the realization of the situation presented in N and the realization of the situation presented in S
Locus of the effect:	N and S

INTERPRETATION	
Constraints on the Nucleus:	None
Constraints on the Satellite:	None
Constraints on N+S combination:	S relates the situation presented in N to a framework of ideas not involved in N itself and not concerned with W's positive regard
The effect:	R recognizes that S relates the situation presented in N to a framework of ideas not involved in the knowledge presented in N itself
Locus of the effect:	N and S
EVALUATION	
Constraints on the Nucleus:	None
Constraints on the Satellite:	None
Constraints on N+S combination:	S relates the situation in N to the degree of W's positive regard toward the situation presented in N
The effect:	R recognizes that the situation presented in S assesses the situation presented in N and recognizes the value it assigns
Locus of the effect:	N and S
RESTATEMENT	
Constraints on the Nucleus:	None
Constraints on the Satellite:	None
Constraints on N+S combination:	S restates N, where S and N are of comparable bulk
The effect:	R recognizes S as a restatement of N
Locus of the effect:	N and S
SUMMARY	
Constraints on the Nucleus:	N must be more than one unit
Constraints on the Satellite:	None
Constraints on N+S combination:	S presents a restatement of the content of N, that is shorter in bulk
The effect:	R recognizes S as a shorter restatement of N
Locus of the effect:	N and S

A.1.2 Paratactic relationships

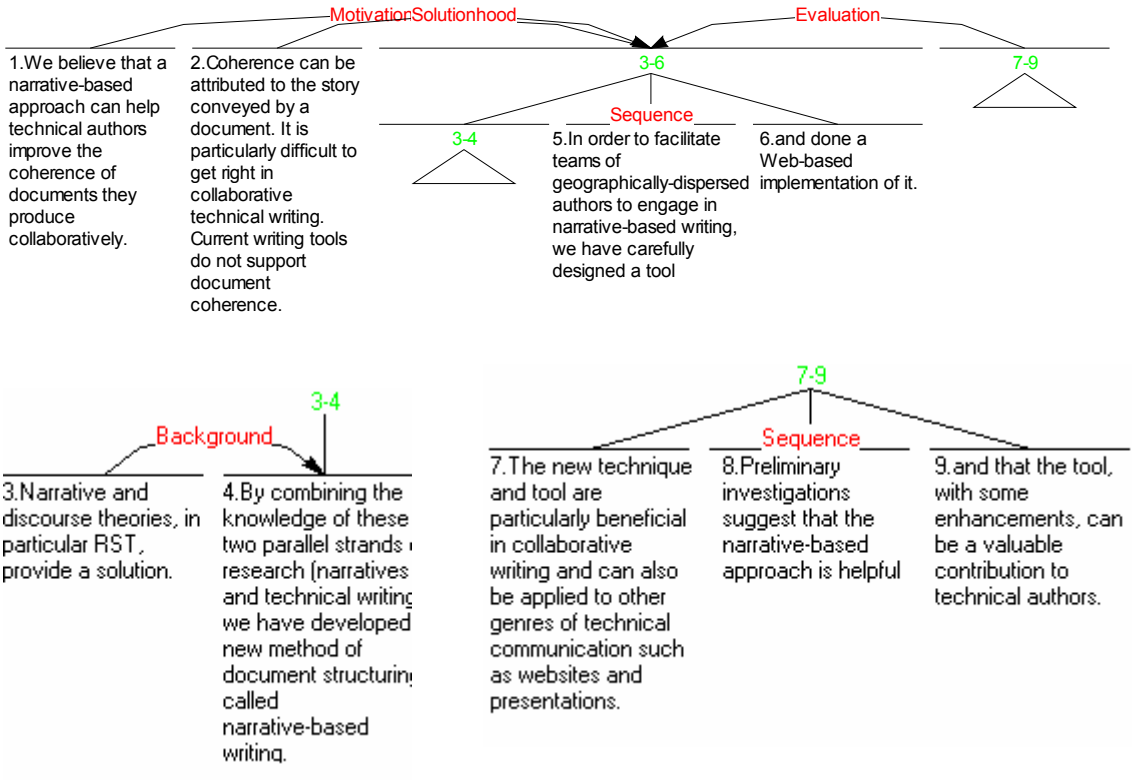
SEQUENCE	
Constraints on the Nucleus:	Multi-nuclear
Constraints on the combination of nuclei:	A succession relationship between the situations is presented in the nuclei
The effect:	R recognizes the succession relationship among the nuclei
Locus of the effect:	Multiple nuclei
CONTRAST	
Constraints on the Nucleus:	Multi-nuclear
Constraints on the combination of nuclei:	No more than two nuclei; the situations presented in these two nuclei are (a) comprehended as the same in many respects, (b) comprehended as differing in a few respects and (c) compared with respect to one or more of these differences
The effect:	R recognizes the comparability and the difference(s) yielded by the comparison being made
Locus of the effect:	Multiple nuclei

A.2 RST analyses of the DNs in the thesis

The RS-trees for the DNs that were not analysed in the main body of this thesis are presented here. Note that more than one RST analysis is possible for a given text. The important point for coherence is whether or not the RST relationships can be assembled into a tree structure. The DN for each chapter in the thesis is analysed below. The first and last chapters contained the DN for the entire thesis. The DN for chapter 3 was analysed in chapter 4. The rest of the DNs are analysed below.

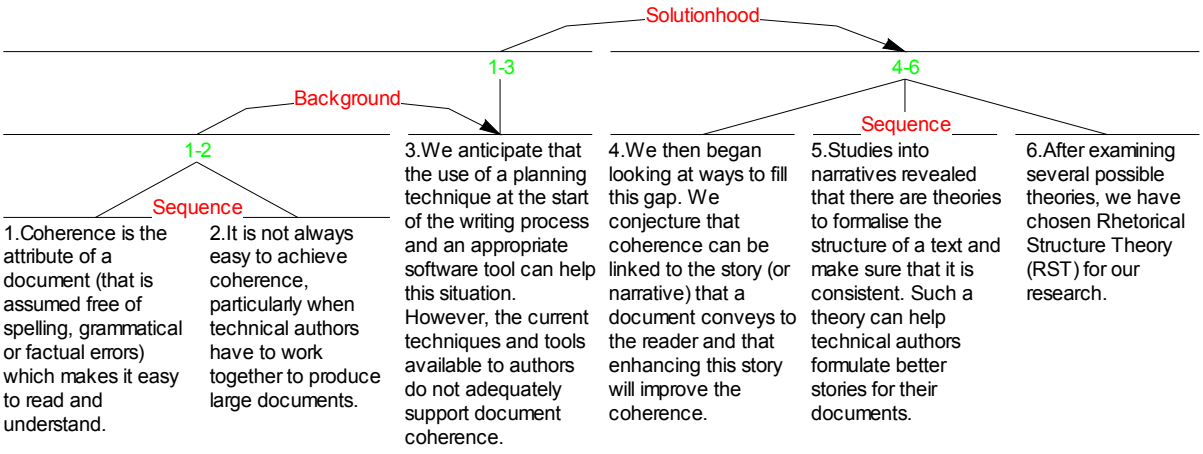
A.2.1 Chapter 1 and 9 – DN for the thesis

[We believe that a narrative-based approach can help technical authors improve the coherence of documents they produce collaboratively.][1] [Coherence can be attributed to the story conveyed by a document. It is particularly difficult to get right in collaborative technical writing. Current writing tools do not support document coherence.][2] [Narrative and discourse theories, in particular RST, provide a solution.][3] [By combining the knowledge of these two parallel strands of research (narratives and technical writing), we have developed a new method of document structuring called narrative-based writing.][4] [In order to facilitate teams of geographically-dispersed authors to engage in narrative-based writing, we have carefully designed a tool][5] [and done a Web-based implementation of it.][6] [The new technique and tool are particularly beneficial in collaborative writing and can also be applied to other genres of technical communication such as websites and presentations.][7] [Preliminary investigations suggest that the narrative-based approach is helpful][8] [and that the tool, with some enhancements, can be a valuable contribution to technical authors.][9]



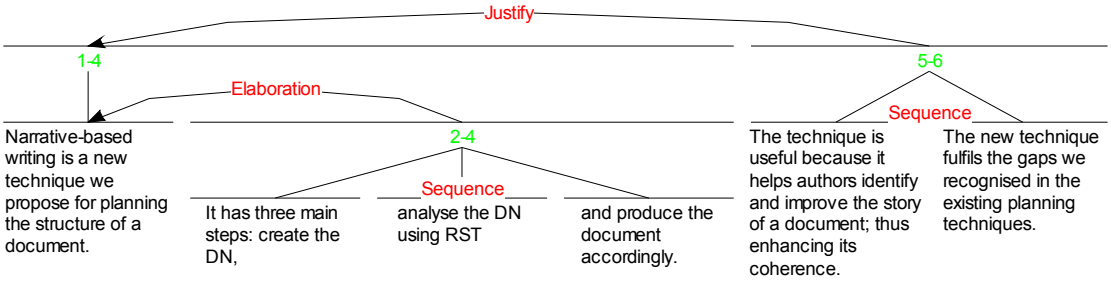
A.2.2 Chapter 2 DN

[Coherence is the attribute of a document (that is assumed free of spelling, grammatical or factual errors) which makes it easy to read and understand.][1] [It is not always easy to achieve coherence, particularly when technical authors have to work together to produce large documents.][2] [We anticipate that the use of a planning technique at the start of the writing process and an appropriate software tool can help this situation. However, the current techniques and tools available to authors do not adequately support document coherence.][3] [We then began looking at ways to fill this gap. We conjecture that coherence can be linked to the story (or narrative) that a document conveys to the reader and that enhancing this story will improve the coherence.][4] [Studies into narratives revealed that there are theories to formalise the structure of a text and make sure that it is consistent. Such a theory can help technical authors formulate better stories for their documents.][5] [After examining several possible theories, we have chosen Rhetorical Structure Theory (RST) for our research.][6]



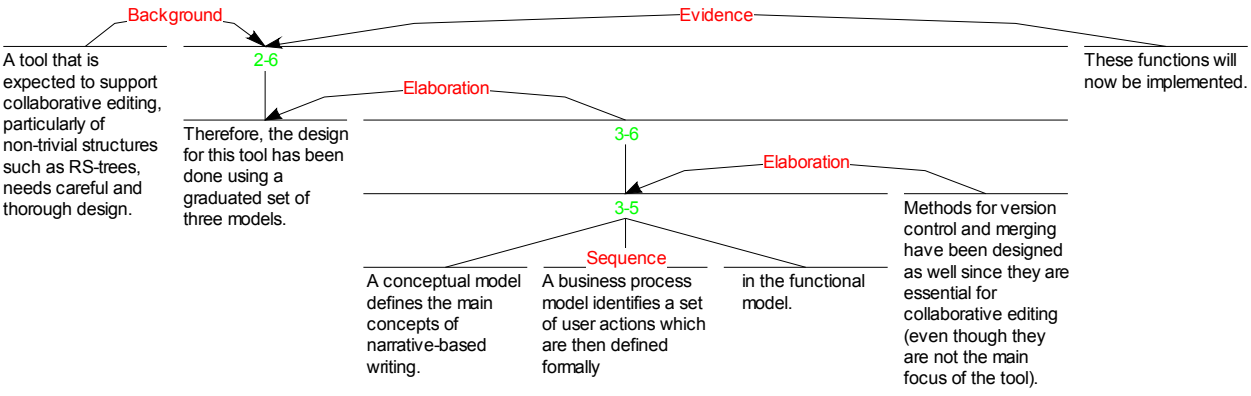
A.2.3 Chapter 4 DN

[Narrative-based writing is a new technique we propose for planning the structure of a document.][1] [It has three main steps: create the DN,][2] [analyse the DN using RST][3] [and produce the document accordingly.][4] [The technique is useful because it helps authors identify and improve the story of a document; thus enhancing its coherence.][5] [The new technique fulfils the gaps we recognised in the existing planning techniques.][6]



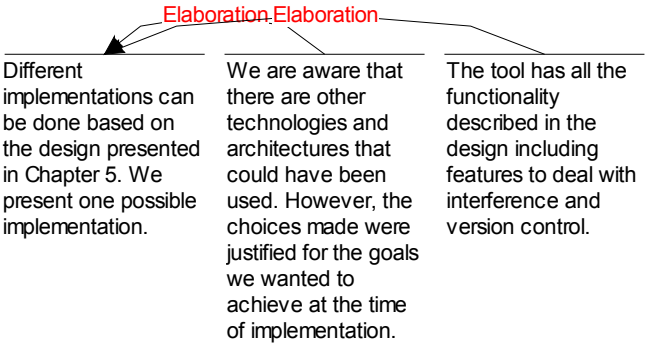
A.2.4 Chapter 5 DN

[A tool that is expected to support collaborative editing, particularly of non-trivial structures such as RS-trees, needs careful and thorough design.][1] [Therefore, the design for this tool has been done using a graduated set of three models.][2] [A conceptual model defines the main concepts of narrative-based writing.][3] [A business process model identifies a set of user actions which are then defined formally][4] [in the functional model.][5] [Methods for version control and merging have been designed as well since they are essential for collaborative editing (even though they are not the main focus of the tool).][6] [These functions will now be implemented.][7]



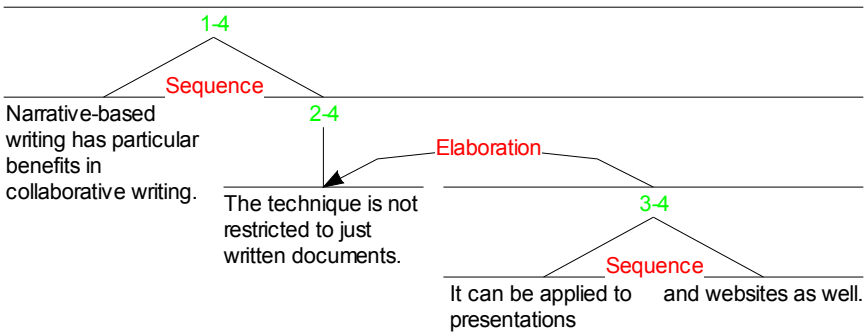
A.2.5 Chapter 6 DN

[Different implementations can be done based on the design presented in Chapter 5. We present one possible implementation.][1] [We are aware that there are other technologies and architectures that could have been used. However, the choices made were justified for the goals we wanted to achieve at the time of implementation.][2] [The tool has all the functionality described in the design including features to deal with interference and version control.][3]



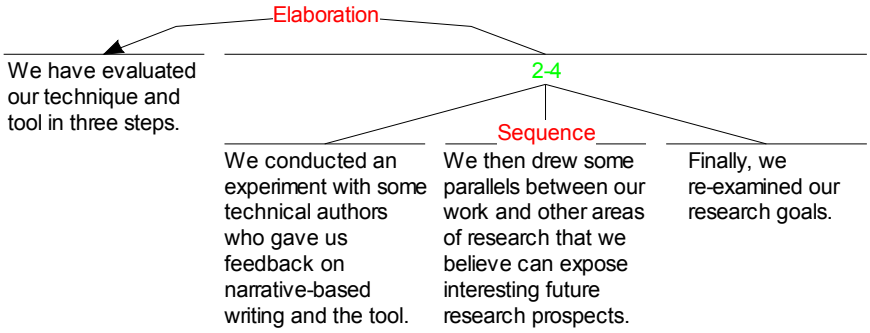
A.2.6 Chapter 7 DN

[Narrative-based writing has particular benefits in collaborative writing.][1] [The technique is not restricted to just written documents.][2] [It can be applied to presentations][3] [and websites as well.][4]



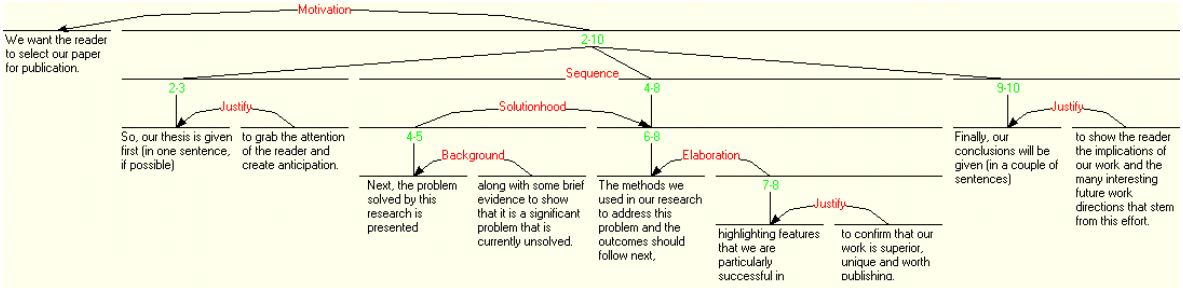
A.2.7 Chapter 8 DN

[We have evaluated our technique and tool in three steps.]1 [We conducted an experiment with some technical authors who gave us feedback on narrative-based writing and the tool.]2 [We then drew some parallels between our work and other areas of research that we believe can expose interesting future research prospects.]3 [Finally, we re-examined our research goals.]4



A.3 DN for an abstract of a research paper

[We want the reader to select our paper for publication.]1 [So, our thesis is given first (in one sentence, if possible)]2 [to grab the attention of the reader and create anticipation.]3 [Next, the problem solved by this research is presented]4 [along with some brief evidence to show that it is a significant problem that is currently unsolved.]5 [The methods we used in our research to address this problem and the outcomes should follow next,]6 [highlighting features that we are particularly successful in]7 [to confirm that our work is superior, unique and worth publishing.]8 [Finally, our conclusions will be given (in a couple of sentences)]9 [to show the reader the implications of our work and the many interesting future work directions that stem from this effort.]10



Appendix B

Implementations

This appendix contains an outline of two previous prototypes we implemented and a listing of the Java methods of the current tool.

B.1 Previous prototypes

A few prototypes were built before the current tool to explore the use of different technologies and study the ways in which document coherence could be supported. The two most relevant prototypes are briefly described below.

B.1.1 Narrative support for research proposals

The first prototype provided, in essence, a template for research proposals. The users were prompted for a descriptive answer and a key sentence in response to twelve questions. The questions were:

1. What is the description and significance of the problem?
2. What are the previous attempts to solve this problem?
3. What is my/our attempt to solve this?
4. Alternative approaches considered?
5. What exactly will we do?
6. What are the results we hope to achieve/have achieved?
7. Who will do these tasks?
8. Why are they qualified to do these tasks?
9. What equipment/software will we need?
10. How much will they cost?
11. Total cost (direct and indirect)?
12. Total time needed?

The answers were stored in an XML file (sample below) and used to generate a research proposal, an abstract and an executive summary using XSLT stylesheets. The 'SEQ' attribute of the 'PART' element in the XML provided a unique identifier for each part of the proposal (see below).

```
<RESEARCH_PROPOSAL Title="Finding the perfect programming language">
  <PART QUESTION="What exactly will we do?" SEQ="1">
     <TEXT> Descriptive answer </TEXT>
     <KEY_SENTENCE> … </KEY_SENTENCE>
  </PART>
  …
</RESEARCH_PROPOSAL>
```

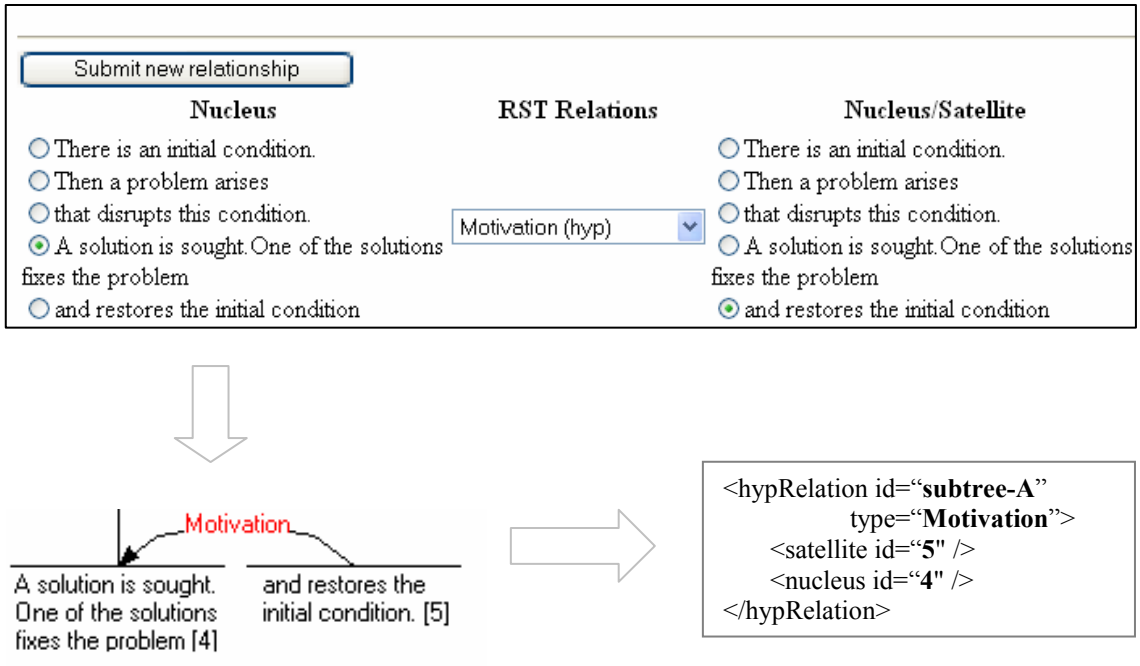
The answers were assembled in specific ways (hard coded into the tool) to produce a research proposal, an abstract and an executive summary. For example, the Introduction of the proposal was composed of the following:

Key sentence of problem description
Context and deeper explanation of problem
Key sentence about expected results
Key sentence about literature review
Key sentence of methodology

The generated abstract was a sequence of the key sentences glued together by phrases such as 'We hope to achieve' and 'We estimate this will cost'. The executive summary contained key sentences of the problem and method, followed by detailed accounts of the cost and time.

B.1.2 CANS (Computer-Aided Narrative Support)

We quickly moved away from a strong coupling between the structure of a document and its content. CANS was a single-author tool that allowed authors to create a DN and analyse it using RST (De-Silva and Henderson, 2005). The users accessed the tool via an HTML interface and the RST structures were stored using URML (Underspecified Rhetorical Markup Language) (Reitter and Stede, 2003b, Reitter and Stede, 2003a). URML is an XML format suggested for storing underspecified RS-trees. An example is shown below.



The author had to link each segment of the DN to a corresponding section in the document. This was done by specifying a question that will later prompt the author for the content of that section. The questions had to be entered when the DN was first created. These narrative structures and questions were stored in the tool.

When an author wanted to create a document, he would choose the appropriate DN for it and then answer the questions that were relevant to that DN. These answers were then ordered according to the sequence of segments in the DN. The tool could also traverse the RS-tree in a different way (using the recommended satellite and nucleus ordering suggested by Mann and Thompson) and suggest an alternate narrative for the document. This feature of being able to explore alternative structures for a document was a key feature of this tool.

The functionality was provided by JSP and XSLT. They were contained in a J2EE server. The XML data was stored in flat files. Later, we used Xindice (http://xml.apache.org/xindice/) to maintain the XML files which made the code to access and edit the data much shorter and easier.

B.2 List of Java methods in the current tool

This section lists the Java methods that were used to implement the tool. We first present a few of the "housekeeping" methods that were needed to access and update the database, and increment unique indices. Then we present the methods that correspond to the functions designed in Chapter 5: the six core functions and the larger functions that implement user actions.

B.2.1 Housekeeping methods

(1) The method below returns a connection to the database (db2).

```
public Connection getConnection(){
     String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
     String url = "jdbc:odbc:db2";
     String username = "", password = "";
     Connection connection = null;
     try {
          Class.forName(driver);
          connection =
          DriverManager.getConnection(url,username,password);
     }catch (ClassNotFoundException cnfe){
          System.err.println("Error loading driver: " + cnfe);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
          return connection;
}
```

(2) The method below closes the connection to the database.

```
public void close(Connection connection){
     try {
          connection.close();
     }catch (Exception e){
          System.err.println("Error closing the connection: " + e);
     }
}
```

(3) The methods below extract a particular field from the database. The arguments to the methods provide the details necessary for the SQL query: the name of the field that needs to be retrieved, the name of the table, the name of the field that needs to be compared and the value it has to be compared to. The SQL queries in this situation are expected to return just one field.

The method below extracts a field from the database which is of type String.

```
public String getStringField(String field_retrieve, String table,
                             String field_compare, int id){
     String field = "";
     try {
          Connection con = this.getConnection();
          //form and execute SQL query
          String query = "SELECT " + field_retrieve + " FROM " +
                   table + " WHERE " + field_compare + "=" + id;
          Statement statement = con.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
          while (resultSet.next()){ //Should only have one element
               field = resultSet.getString(field_retrieve);
          }
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
     return field; //Return string field
}
```

The method below extracts a field from the database which is of type int.

```
public int getIntField(String field_retrieve, String table,
                       String field_compare, int id){
     int field = -2; //Returns -2 if there is no other value
     try {
          Connection con = this.getConnection();
          //form and execute SQL query
          String query = "SELECT " + field_retrieve + " FROM " +
               table + " WHERE " + field_compare + "=" + id;
          Statement statement = con.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
          while (resultSet.next()){ //Should only have one element
               field = resultSet.getInt(field_retrieve);
          }
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
     return field; //return int field
}
```

These two methods could have been combined into one generic method that retrieves a field of a specified type. The method would then need to return an object of type Vector or something like that. This would have, however, lengthened the processing to extract the required information in the method that calls this method. Therefore, we have decided to have two separate methods.

(4) The method below inserts new records to the database. It takes a SQL INSERT statement as an argument and executes it. (There will be no SQL UPDATE statements since we do not update any information in-situ in the database.)

```
public void put(String stmt){
     try {
          Connection con = this.getConnection();
          Statement statement = con.createStatement();
          statement.executeUpdate(stmt);
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
}
```

(5) The method below generates the IDs for nodes and RS-trees. The method reads the current value of the ID, increments it by one and stores the new ID back in the table. The new ID is also returned by the method. The "autonumbering" feature of Microsoft Access could have been used to achieve some of this functionality. The reasons against doing that were presented in Chapter 6.

```
public int newID(String field){
     System.out.println("Generating new id for " + field);
     int new_id = -1;
     try{
          Connection con = this.getConnection();
          //Get current ID from INDEX table
          String query = "SELECT " + field + " FROM INDEX";
          Statement statement = con.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
          resultSet.next();
          new_id = resultSet.getInt(field) + 1; //Increment by 1
          //Store new ID back in table
          String update = "UPDATE INDEX SET " + field + "=" + new_id;
          statement.executeUpdate(update);
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
     return new_id;
}
```

(6) The method below assigns new version numbers for a given RS-tree. The process is identical to the method above except that the following method uses the INDEX2 table (which contains the latest version number for each RS-tree in the database).

```
public synchronized int newVersion(int doc){
     int new_id = -1;
     try{
          Connection con = this.getConnection();
          String query = "SELECT Version FROM Index2 WHERE
                                                  Doc_ID=" + doc;
          Statement statement = con.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
          resultSet.next();
          new_id = resultSet.getInt("Version") + 1; //Increment by 1
          String update = "UPDATE INDEX2 SET Version=" + new_id +
               " WHERE Doc_ID=" + doc;   //Write new id back to table
          statement.executeUpdate(update);
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error: " + sqle);
     }catch (InterruptedException ie){
          System.err.println("Error: " + ie);
     }
     return new_id;
}
```

B.2.2 Methods corresponding to the six core functions in Chapter 5

(1) The method below corresponds to the `getChildren(n)` function. It returns a vector containing the set of immediate children of node n. In the implementation, the functionality to order the children was separated into a different method since the children do not always need to be ordered. For most functions, a set of unordered children is sufficient. The nodes only need to be ordered for a few of the other methods.

```
getChildren(n){
   children = new set(Node)
   pcrelations = all PC relations
                 where n is a parent
   for (each parent,child in pcrelations){
      children.add(child)
   }
   ## order children (separate function below)
   return children
}
```

```
public Vector getChildren(int n){
   Vector children = new Vector();
   try {
      Connection con = this.getConnection();
      String query = "SELECT Child FROM PCRel WHERE Parent=" + n;
      Statement statement = con.createStatement();
      ResultSet resultSet = statement.executeQuery(query);
      while (resultSet.next()){ //Add each child to vector
         children.addElement(new Integer(resultSet.getInt("Child")));
      }
      this.close(con);
   }catch (SQLException sqle){
      System.err.println("Error with connection: " + sqle);
   }
   return children;
}
```

The method to order the children is below. It is in a dashed box because it is a part of the getChildren() function that was discussed in Chapter 5 (even though it is a separate method in the actual Java).

```
public Vector orderChildren(Vector children){
     Vector orderedChildren = new Vector();
     //Get NXT relationships for these children
     Vector nxtrels = this.getNXTRelationships(children);
     //Find first
     int firstchild = -1;
     for(int i=0; i<nxtrels.size(); i++){
          //If first node is -1 i.e. (-1,X), then X is first
```

```
          if ( ((((Integer)((Vector)nxtrels.elementAt(i))
                              .elementAt(0)).intValue()) < 0) ){

               firstchild = ((Integer)((Vector)nxtrels.elementAt(i))
                                   .elementAt(1)).intValue();
          }
     }
     //If first child has been located successfully
     if (firstchild > 0) {
          orderedChildren.addElement(new Integer(firstchild));
          //Navigate the NXT relationships for the rest
          int nextchild=this.getIntField("Second", "NXTRel", "First",
                                                        firstchild);
          while (nextchild!=-2){ //getIntField returns -2 if error
               orderedChildren.addElement(new Integer(nextchild));
               nextchild = this.getIntField("Second", "NXTRel",
                                                   "First", nextchild);
          }
     }
     return orderedChildren;
}
```

(2) The method below corresponds to the `getNodes(r)` function. It returns a vector containing all the nodes in a given tree.

```
getNodes(r){
   descendants = new set(Node)
   descendants.add(r)
   children = getChildren(r)
   for(each node in children){
      descendants.add(getNodes(node))
   }
   return descendants
}
```

```
public Vector getNodes(int t){
     Vector nodes = new Vector();
     nodes.addElement(new Integer(t));
     Vector children = this.getChildren(t);
     for(int i=0;i<children.size();i++){
          nodes.addAll(this.getNodes(((Integer)children.elementAt(i))
               .intValue()));
     }
     return nodes;
}
```

(3) The method below corresponds to the contains(n,r) function. It returns true if node n is contained anywhere in tree t.

```
contains(n,r){
     return (n is in getNodes(r))
}
```

```
public boolean contains(int n, int t){
     Vector nodes = this.getNodes(t);
     if (nodes.contains(new Integer(n))){
          return true;
     }
     return false;
}
```

(4) The method corresponds to the function `locate(n,r)` which returns the immediate subtree within tree r that contains node n.

```
locate(n,r){
     children = getChildren(r)
     for(each node in children){
         if (contains(n,node)){
             return node.ID
         }
     return -1
}
```

```
public int locate (int n, int t){
     Vector children = this.getChildren(t);
     for(int i=0;i<children.size();i++){
          if (this.contains(n,
                    (((Integer)children.elementAt(i)).intValue()))){
               return (((Integer)children.elementAt(i)).intValue());
          }
     }
     return -1;
}
```

(5) The method below corresponds to the `getRSTRelationships(r)` function. It returns a vector with all the RST relationships pertaining to the tree with root r.

```
getRSTRelationships(r){
    rels = new set(RST Relationship)
    for(each RST relationship){
      if(contains(satellite,r) AND
                        contains(nucleus,r))
          rels.add(the RST relationship)
    }
    return rels
}
```

```
public Vector getRSTRelationships(int r){
     Vector rels = new Vector();
     Vector nodes = this.getNodes(r);
     try {
          Connection con = this.getConnection();
          //Get all RST relationships in tree with root r
          String query = "SELECT * FROM RSTRel";
          Statement statement = con.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
          while (resultSet.next()){
               int node1 = resultSet.getInt("Node1");
               int node2 = resultSet.getInt("Node2");
               //If both nodes in this version
               if (nodes.contains(new Integer(node1)) &&
                              nodes.contains(new Integer(node2))){
                    Vector entry = new Vector();
                    entry.add(new Integer(node1)); //Node1
                    entry.add(new Integer(node2)); //Node2
                    entry.add(new Integer
                    (resultSet.getInt("Relation_ID"))); //Relation_ID
                    entry.add(new Boolean(resultSet
                    .getBoolean("State"))); //Checked or unchecked
                    rels.add(entry);
               }
          }
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
     return rels;
}
```

(6) The method below corresponds to the getNXTRelationships(children) function.

```
getNXTRelationships(children){
    nxtrels = new set(NXT Relationship)
    for(each NXT Relationship in repository){
      if( (first is in children OR is _ ) AND
          (second is in children)
           nxtrels.add(the NXT relationship)
    }
    return nxtrels
}
```

```
public Vector getNXTRelationships(Vector children){
     Vector nxtrels = new Vector();
     try{
          Connection con = this.getConnection();
          //Get all the NXT relationships
          String query = "SELECT First,Second FROM NXTRel";
          Statement statement = con.createStatement();
          ResultSet resultSet = statement.executeQuery(query);
          while (resultSet.next()){ //For each NXT relationship
               int first = resultSet.getInt("First");
               int second = resultSet.getInt("Second");
               //If first and second nodes are elements in children
               if((children.contains(new Integer(first)) || first==-1)
                       && (children.contains(new Integer(second)))){
                     Vector entry = new Vector();
                     entry.addElement(new Integer(first));
                     entry.addElement(new Integer(second));
                     nxtrels.add(entry); //Add NXT Relationship
               }
          }
          this.close(con);
     }catch (SQLException sqle){
          System.err.println("Error with connection: " + sqle);
     }
     return nxtrels;
}
```

B.2.3 Methods corresponding to the other functions in Chapter 5

(1) The pseudocode for the function `print(c,r)` is given below.

```
print(c, r){
    display c.ID and c.text
    relationships = getRSTRelationships(r)
    for (each rel in relationships){
        if(n==rel.nucleus)
            display details of rel
    }
    children = getChildren(c)
    for (each node in children)
        print(node,r)
}
```

This function has been implemented using two Java functions. The first function gets the RS-tree ID and version number from the user, displays some initial information about the RS-tree and calls the second function.

```
public String read(String author, Integer ver, Integer rstree){
     this.recordLog("Reading document " + rstree.toString() +
               " version " + ver.toString() +" (" + author + ")");
     String document = "";
     document = document.concat("<B>");
     document = document.concat("Version " + ver.intValue() +
               "<BR />"); //Version of the document
     document = document.concat("Title: " +
          this.getTitle(ver.intValue(),rstree.intValue())); //Title
     document = document.concat("</B><BR />");
     int root_node = this.getRoot(ver.intValue(),rstree.intValue());
     document = document.concat("Root Node: " + root_node +
                                                    "<BR />");
     document = document.concat(this.read(root_node, root_node,"
                                                              "));
     return document;
}
```

The second function traverses the RS-tree, displays the node and RST information. The extra "indent" argument is necessary to display the text with proper indentation so as to make the tree structure obvious.

```
public String read (int n, int r, String indent){
     String document = "";
     String line = "";
     Vector children = this.getChildren(n);
     children = this.orderChildren(children); //order the children
     for(int i=0;i<children.size();i++){ //For each child
          int current_node =
                    ((Integer)children.elementAt(i)).intValue();
          line = indent;
          line = line.concat("<FONT COLOR=gray><I>(" + current_node +
                         ") </I></FONT>"); //Node ID in italics
          line = line.concat(this.getStringField("Text","Node","ID",
                              current_node)); //Text of the node
          line = line.concat("<BR />"); //New line
          document = document.concat(line);
          Vector relations = this.getRSTRelationships(n);
          for(int ii=0;ii<relations.size();ii++){ //For each rel
               int node1 = ((Integer)((Vector)relations.elementAt(ii))
                                        .elementAt(0)).intValue();

               int node2 = ((Integer)((Vector)relations.elementAt(ii))
                                        .elementAt(1)).intValue();
               int rel_id = (Integer)((Vector)relations.elementAt(ii))
                                        .elementAt(2)).intValue();
               boolean checked = ((Boolean)((Vector)relations
                         .elementAt(ii)).elementAt(3)).booleanValue();
               if (node1==current_node){ //If node1 is current node
                     String font = "blue"; //Satisfied → blue
                     if(checked==false){ // Unsatisfied → red
                          font = "red";
                     }
                     line = indent;
                     //Create the HTML
                     line = line.concat("<font color=" + font +
                          ">Related by a "+ this.getStringField
                          ("Name","Relation","ID",rel_id) + "
                     relationship with node " + node2 + "</font>");
                     line = line.concat("<BR />");
                     document = document.concat(line);
               }
          }
          document = document.concat(
               this.read(current_node,r,indent.concat("    ")));
     }
     return document;
}
```

(2) The method below corresponds to the `addnode(...)` function.

```
addNode(n,p,s,c,r){
   1. Replicate the text of node c in a new node (say, newnode)
   2. Traverse the tree until the parent node p is found
      (a) if (c = = p)
             Add a new node with text s (say, newnode2)
             Store PC (newnode.ID, newnode2.ID)
             Store appropriate NXT relationships (see a.1 below)
      (b) else
             x   = locate (p,c)
             xx = addNode (n,p,s,x,r)
   3. Fix relationships
   4. Return ID of newnode
}
```

The sections in the Java that relate to the pseudocode above have been grouped (indicated by boxes around them). A second method receives the input from the user, calls the method below and adds a new version of the RS-tree into the database.

```
public int addNode (int n, int p, String s, int c, int r){
     int newnode = this.newID("Node_ID");
     //Replicate the text of node c in newnode
     this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
              this.getStringField("Text","Node","ID",c) + "')" );
     int x = -10, xx = -10, newnode3 = -10, nextofn = -10;
     nextofn = this.getIntField("Second", "NXTRel", "First", n);
     if (c==p){ //If this is the parent
          //Adding the new child
          int newnode2 = this.newID("Node_ID");
          this.put("INSERT INTO Node VALUES (" + newnode2 + ",'" +
                                (s.replaceAll("'","''")) + "')" );
          //Add relevant PC and NXT relationships
          this.put("INSERT INTO PCRel VALUES (" + newnode + "," +
                                                 newnode2 + ")");
          this.put("INSERT INTO NXTRel VALUES (" + n + "," +
                                                 newnode2 +")" );
          //If n has a next (in original sequence)
          if (nextofn != -2){
               //Replace nextofn with a newnode3
               newnode3 = this.newID("Node_ID");
               this.put("INSERT INTO Node VALUES (" + newnode3 + ",'"
          + this.getStringField("Text","Node","ID",nextofn) + "')" );
               this.put("INSERT INTO NXTRel VALUES (" + newnode2 + ","
                                                 + newnode3 +")" );
               //Similarly, check if nextofn has a next
               if((this.getIntField("Second", "NXTRel", "First",
                                                 nextofn))!= -2){
                    this.put("INSERT INTO NXTRel VALUES (" + newnode3
```

```
+ "," + this.getIntField("Second", "NXTRel", "First", nextofn) +")"
);
                }
            }
        }else{ //Else, recurse
            x = this.locate(p,c);
            xx = this.addNode(n, p, s, x, r);
        }
        /*Fix other relationships (replace x with xx, c with newnode and
        n with newnode3) */
        //PC relationships
        Vector children = this.getChildren(c); //No need to order
        for(int i=0;i<children.size();i++){
            if((((Integer)children.elementAt(i)).intValue())==x){
            // (newnode,xx)
            //Need to sort x since the recursion stops at the parent
            this.put("INSERT INTO PCRel VALUES (" + newnode + "," + xx +
")");
            }else if(nextofn > 0 &&
             (((Integer)children.elementAt(i)).intValue())==nextofn){
            // (newnode,newnode3)
                this.put("INSERT INTO PCRel VALUES (" + newnode + "," +
newnode3 + ")");
            }else{ // (newnode, child)
                this.put("INSERT INTO PCRel VALUES (" + newnode + "," +
((Integer)children.elementAt(i)).intValue() + ")");
            }
        }
        //NXT relationships
        Vector nxtrels = getNXTRelationships(children);
        for(int i=0;i<nxtrels.size();i++){ //For each NXT relationship
            int first =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(0)).intValue());
            int second =
(((Integer)((Vector)nxtrels.elementAt(i)).elementAt(1)).intValue());
            if(first == x){ //replace x with xx
                this.put("INSERT INTO NXTRel VALUES (" + xx + "," +
second + ")");
            }else if (second == x){
                this.put("INSERT INTO NXTRel VALUES (" + first + "," +
xx + ")");
            }
            //NXT relationships pertaining to nextofn have already been
resolved
        }
        //RST relationships
        Vector relations = this.getRSTRelationships(r);
        for(int ii=0;ii<relations.size();ii++){
                int node1 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(0)).intValue();
                int node2 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(1)).intValue();
                int rel_id =
```

```
((Integer)((Vector)relations.elementAt(ii)).elementAt(2)).intValue();
              boolean checked = false; //All become unchecked
              if(c==node1){
                    this.put("INSERT INTO RSTRel VALUES (" + newnode
+ "," + node2 + "," + rel_id + "," + checked + ")" );
              }else if (c==node2){
                    this.put("INSERT INTO RSTRel VALUES (" + node1 +
"," + newnode + "," + rel_id + "," + checked + ")" );
              }
              if(nextofn==node1){
     this.put("INSERT INTO RSTRel VALUES (" + newnode3 + "," + node2
+ "," + rel_id + "," + checked + ")" );
              }else if (nextofn==node2){
     this.put("INSERT INTO RSTRel VALUES (" + node1 + "," + newnode3
+ "," + rel_id + "," + checked + ")" );
              }
              if(x==node1){
                    this.put("INSERT INTO RSTRel VALUES (" + xx + ","
+ node2 + "," + rel_id + "," + checked + ")" );
              }else if (x==node2){
                    this.put("INSERT INTO RSTRel VALUES (" + node1 +
"," + xx + "," + rel_id + "," + checked + ")" );
              }
     }

     return newnode;
}
```

(3) The method below corresponds to the `replaceNode(...)` function.

```
replaceNode(n,s,c,r){
     1. Generate a new node with a unique ID (say, newnode)
     2. Traverse the tree until the specified node is found
          (a) if (c = = n)
               Store newnode (containing text s)
          (b) else
               x   = locate (n,c)
               xx = replaceNode (n,s,x,r)
               Store the contents of node c in a new node (newnode)
               Fix PC relationships
     3. Fix other relationships (NXT and RST)
     4. Return ID of newnode
}
```

```
public int replace (int n, String s, int c, int r){
     int newnode = this.newID("Node_ID"); //Generate new Node ID
     if(c==n){
          //Store a new node with the new content
          this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
                                  (s.replaceAll("'","''")) + "')" );
     }else {
          //Replicate current node (in the path)
          this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
               this.getStringField("Text","Node","ID",c) + "')");
          int x = this.locate(n,c);
          int xx = this.replace(n,s,x,r);
          //PC relationships
          Vector children = this.getChildren(c);
          for(int i=0;i<children.size();i++){
               //For each child, set appropriate PC relationships
               if((((Integer)children.elementAt(i)).intValue())==x){
                     this.put("INSERT INTO PCRel VALUES (" + newnode +
                                                 "," + xx + ")");
               }else{ // (newnode, child)
                     this.put("INSERT INTO PCRel VALUES (" + newnode +
          "," + ((Integer)children.elementAt(i)).intValue() + ")");
               }
          }
     }
     //NXT Relationships
     Vector nxtrels = getNXTRelationships(this.getNodes(r));
     for(int i=0;i<nxtrels.size();i++){ //For each NXT relationship
          int first =
(((Integer)((Vector)nxtrels.elementAt(i)).elementAt(0)).intValue());
          int second =
(((Integer)((Vector)nxtrels.elementAt(i)).elementAt(1)).intValue());
          if(first == c){ //replace c with newnode
               this.put("INSERT INTO NXTRel VALUES (" + newnode + ","
+ second + ")");
          }else if (second == c){
               this.put("INSERT INTO NXTRel VALUES (" + first + "," +
newnode + ")");
          }
     }
     //RST Relationships
     Vector relations = this.getRSTRelationships(r);
     for(int ii=0;ii<relations.size();ii++){
          int node1 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(0)).intValue();
          int node2 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(1)).intValue();
          int rel_id =
((Integer)((Vector)relations.elementAt(ii)).elementAt(2)).intValue();
          boolean checked = false;
          if(c==node1){
               this.put("INSERT INTO RSTRel VALUES (" + newnode + ","
                     + node2 + "," + rel_id + "," + checked + ")" );
```

```
          }else if (c==node2){
               this.put("INSERT INTO RSTRel VALUES (" + node1 + "," +
                       newnode + "," + rel_id + "," + checked + ")" );
          }
     }
     return newnode;
}
```

A second method receives the input from the user, calls the method above and adds a new version of the RS-tree into the database.

(4) The method below corresponds to the `removeNode(...)` function.

```
removeNode(n,c,r){
     1. Generate a new node with a unique ID (say, newnode)
     2. Traverse the tree until the specified node is found
          (a) if (c == n)
                Replace the node just after n (newnode2)
                 Set relevant NXT relationships
          (b) else
               x   = locate (n,c)
               xx = removeNode (n,x,r)
               Replicate the content of node c in newnode
     3. Fix relationships (discussed below)
     4. Return ID of newnode
}
```

```
public int removeNode (int n, int c, int r){
     int newnode = this.newID("Node_ID");
     int nextofn = this.getIntField("Second", "NXTRel", "First", n);
     int x = -10, xx = -10, newnode2 = -10; //Initialising these.
     if (c==n){
          //Insert a new node to replace the node just after
          if(nextofn != -2){
               newnode2 = this.newID("Node_ID");
               this.put("INSERT INTO Node VALUES (" + newnode2 + ",'"
+ this.getStringField("Text","Node","ID",nextofn) + "')" );
               //Fix the affected NXT relationships
               //Check if n had a node before it in the sequence
               if (this.getIntField("First", "NXTRel", "Second", n) !=
-2){
                    this.put("INSERT   INTO   NXTRel   VALUES   ("   +
this.getIntField("First", "NXTRel", "Second", n) + "," + newnode2
+")" );

               }
               //Similarly, check if nextofn has a next
               if((this.getIntField("Second",    "NXTRel",    "First",
nextofn))!= -2){
```

```
                        this.put("INSERT INTO NXTRel VALUES (" +
newnode2 + "," + this.getIntField("Second", "NXTRel", "First",
nextofn) +")" );
                }
            }
        }else{
            x = this.locate(n,c);
            xx = this.removeNode(n, x, r);
            this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
this.getStringField("Text","Node","ID",c) + "')" );
        }
        //Fix PC relationships
        Vector children = this.getChildren(c);
        for(int i=0;i<children.size();i++){
            if((((Integer)children.elementAt(i)).intValue())!=n){
            //If it is NOT the child that needs to be deleted
                if((((Integer)children.elementAt(i)).intValue())==x){
                    this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + xx + ")");
                }else
if((((Integer)children.elementAt(i)).intValue())==nextofn){
                    this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + newnode2 + ")");
                }else{ // (newnode, child)
                    this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + ((Integer)children.elementAt(i)).intValue() + ")");
                }
            }
        }
        //Fix NXT relationships
        Vector nxtrels = getNXTRelationships(children);
        for(int i=0;i<nxtrels.size();i++){ //For each NXT relationship
            int first =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(0)).intValue());
            int second =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(1)).intValue());
            if (first !=n && second !=n){
                if(first == c){ //replace c with newnode
                    this.put("INSERT INTO NXTRel VALUES (" + newnode
+ "," + second + ")");
                }else if (second == c){
                    this.put("INSERT INTO NXTRel VALUES (" + first +
"," + newnode + ")");
                }
            } //The NXT relationships involving newnode2 have already
being sorted
        }
        //Fix RST relationships
        Vector relations = this.getRSTRelationships(r); //Get relations
in this tree
        for(int ii=0;ii<relations.size();ii++){
            int node1 =
 (Integer)((Vector)relations.elementAt(ii)).elementAt(0)).intValue();
```

```
         int node2 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(1)).intValue();
         int rel_id =
((Integer)((Vector)relations.elementAt(ii)).elementAt(2)).intValue();
         boolean checked = false;
         if (node1 != n && node2 != n){
              if(node1==nextofn){
                   this.put("INSERT INTO RSTRel VALUES (" + newnode
+ "," + node2 + "," + rel_id + "," + checked + ")" );
              }else if (node2==nextofn){
                   this.put("INSERT INTO RSTRel VALUES (" + node1 +
"," + newnode2 + "," + rel_id + "," + checked + ")" );
              }else     if(node1==c){     //Add     rst     relation
(newnode,node2,rel_id,false)
                   this.put("INSERT INTO RSTRel VALUES (" + newnode
+ "," + node2 + "," + rel_id + "," + checked + ")" );
              }else  if  (node2==c){  //Add  rst  relation  (node1,
newnode, rel_id, false)
                   this.put("INSERT INTO RSTRel VALUES (" + node1 +
"," + newnode + "," + rel_id + "," + checked + ")" );
              }
         }
     }
     return newnode;
}
```

(4) The method below corresponds to the `createSpan(...)` function.

```
createSpan(nodes,p,c,r){
      1. Store the text of node c in a new node (say, newnode)
      2. Traverse the tree until the specified parent p is found
         (a) if (c = = p)
              Create a new node (say, newnode2) ##span - no text
              Store PC (newnode.ID, newnode2.ID)
              Store relevant NXT relationships
              for (each node in nodes)
                 Store a new node for the first node in the span
                 For rest, store PC (newnode2.ID, node.ID
         (b) else
              x   = locate (nodes[1],c)
              xx  = createSpan (nodes,p,x,r)
      3. Fix relationships (discussed below)
      4. Return ID of newnode
}
```

```
public int createSpan (Vector nodes, int p, int c, int r){
     int newnode = this.newID("Node_ID");
     int x=-10, xx=-10, newnode2=-10,newnode3=-10;
     this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
this.getStringField("Text","Node","ID",c) + "')" );
     if (c==p){ //If this is the parent, add the new subtree
          newnode2 = this.newID("Node_ID");
          this.put("INSERT INTO Node VALUES (" + newnode2 + ",'')" );
//blank node, since it will be span
          this.put("INSERT INTO PCRel VALUES (" + newnode + "," +
newnode2 + ")");
          //Adjust specific NXT relationships
          if(this.getIntField("First","NXTRel","Second",
               ((Integer)nodes.elementAt(0)).intValue())!= -2){
               this.put("INSERT INTO NXTRel VALUES ("
this.getIntField("First","NXTRel","Second",((Integer)nodes.elementAt(
0)).intValue()) + "," + newnode2 + ")");
          }
          if(this.getIntField("Second","NXTRel","First",
((Integer)nodes.elementAt(nodes.size()-1)).intValue())!= -2){
               this.put("INSERT INTO NXTRel VALUES (" + newnode2 +
this.getIntField("Second","NXTRel","First",((Integer)nodes.elementAt(
nodes.size()-1)).intValue()) + ")");
          }
          //Process the nodes that need to be in the span
          for(int i=0;i<nodes.size();i++){
               if(i==0){ //the first node is replaced with a new node
                     newnode3 = this.newID("Node_ID");
                     this.put("INSERT INTO Node VALUES (" + newnode3 +
",'"                                                              +
this.getStringField("Text","Node","ID",((Integer)nodes.elementAt(i)).
intValue()) + "')" );
                     this.put("INSERT INTO NXTRel VALUES (" + -1 + ","
+ newnode3 + ")");
                     if(nodes.size() >= 2){
                          this.put("INSERT INTO NXTRel VALUES ("
newnode3 + "," + ((Integer)nodes.elementAt(1)).intValue() + ")");
                     }
                     this.put("INSERT INTO PCRel VALUES (" + newnode2
+ "," + newnode3 + ")");
               Vector                          temp                    =
this.getChildren(((Integer)nodes.elementAt(0)).intValue());
                     for(int ii=0; ii<temp.size(); ii++){
                          this.put("INSERT INTO PCRel VALUES ("
newnode3 + "," + ((Integer)temp.elementAt(ii)).intValue() + ")");
                     }

               }else{ //Just add as a child of newnode2
                     this.put("INSERT INTO PCRel VALUES (" + newnode2
+ "," + ((Integer)nodes.elementAt(i)).intValue() + ")");
               }
          }
```

```
      }else{
         x = this.locate(((Integer)nodes.elementAt(0)).intValue(),c);
       //All values in nodes should have same parent
         xx = this.createSpan(nodes, p, x, r);

      }
      //PC relationships
      Vector children = this.getChildren(c); //No need to order
      for(int i=0;i<children.size();i++){
         if(!nodes.contains(children.elementAt(i))){
              if((((Integer)children.elementAt(i)).intValue())==x){
                    //Need to sort x since the recursion stops at the
parent level
                    this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + xx + ")");
              }else{ // (newnode, child)
                    this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + ((Integer)children.elementAt(i)).intValue() + ")");
              }
         }
      }
      //NXT relationships
      Vector nxtrels = getNXTRelationships(children);
      for(int i=0;i<nxtrels.size();i++){ //For each NXT relationship
         int first =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(0)).intValue());
//Stored to make things easier
         int second =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(1)).intValue());
         if      (!(nodes.contains(new      Integer(first)))      &&
!(nodes.contains(new Integer(second)))){
              if(first == c){ //replace c with newnode
                    this.put("INSERT INTO NXTRel VALUES (" + newnode
+ "," + second + ")");
              }else if (second == c){
                    this.put("INSERT INTO NXTRel VALUES (" + first +
"," + newnode + ")");
              }
         }
      }

      //RST relationships
      Vector  relations  =  this.getRSTRelationships(r);  //Get  RST
relationships in this tree
      for(int ii=0;ii<relations.size();ii++){
         int node1 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(0)).intValue();
         int node2 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(1)).intValue();
         int rel_id =
((Integer)((Vector)relations.elementAt(ii)).elementAt(2)).intValue();
         boolean checked = false; //All relations in the path become
unchecked
```

```
         if((nodes.contains(new          Integer(node1)))          &&
!(nodes.contains(new Integer(node2)))){
             this.put("INSERT INTO RSTRel VALUES (" + newnode2 + ","
+ node2 + "," + rel_id + "," + checked + ")" );
         }else    if    ((nodes.contains(new    Integer(node2)))    &&
!(nodes.contains(new Integer(node1)))){
             this.put("INSERT INTO RSTRel VALUES (" + node1 + "," +
newnode2 + "," + rel_id + "," + checked + ")" );
         }else    if    ((nodes.contains(new    Integer(node1)))    &&
(nodes.contains(new Integer(node2)))){
             if(node1== ((Integer)nodes.elementAt(0)).intValue()){
                   this.put("INSERT INTO RSTRel VALUES (" + newnode3
+ "," + node2 + "," + rel_id + "," + checked + ")" );
             }else                                          if(node2==
((Integer)nodes.elementAt(0)).intValue()){
                   this.put("INSERT INTO RSTRel VALUES (" + node1 +
"," + newnode3 + "," + rel_id + "," + checked + ")" );
             }
         }else       if(x==node1){       //Add       RST       relation
(xx,node2,rel_id,false)
             this.put("INSERT INTO RSTRel VALUES (" + xx + "," +
node2 + "," + rel_id + "," + checked + ")" );
         }else if (x==node2){ //Add RST relation (node1, newnode,
rel_id, false)
             this.put("INSERT INTO RSTRel VALUES (" + node1 + "," +
xx + "," + rel_id + "," + checked + ")" );
         }
      }

      return newnode;
    }
```

(5) The method below corresponds to the addRelationship(…) function.

```
  addRelationship(n1,n2,rel,r,c){
   1. Store the text of node c in a new node (say, newnode)
   2. Traverse the tree until the specified nodes are found
       (a) if (c!=n1 AND c!=n2)
              x1 = locate (n1, c)
              x2 = locate (n2, c)
              xx1 = addRelationship(n1,n2,rel,r,x1)
              if (x1 != x2){
                  xx2 = addRelationship(n1,n2,rel,r,x2)
                  STORE new RST relationship between xx1 and xx2
                  STORE new NXT relationship between xx1 and xx2
                  Fix PC relationships affected by xx1 and xx2

   3. Fix other relationships (similar to replaceNode function)
   4. Return ID of newnode
}
```

The “rel” argument in the Java is an int that is an ID from the RELATION table.

```
public int addRelationship (int n1, int n2, int rel, int r, int c){
  int newnode = this.newID("Node_ID");
  this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
  this.getStringField("Text","Node","ID",c) + "')" );
  int x1=-10, x2=-10, xx1=-10, xx2=-10; //Initialising
  if (c!=n1 && c!=n2){
     x1 = this.locate(n1,c); //n1 and n2 have same parent
     x2 = this.locate(n2,c);
     xx1 = this.addRelationship(n1,n2,rel,r,x1);
     if (x1!=x2){
        xx2 = this.addRelationship(n1,n2,rel,r,x2);
        this.put("INSERT INTO RSTRel VALUES (" + xx1 +
        "," + xx2 + "," + rel + "," + false + ")" );
        // Fix PC relationships
        Vector children = this.getChildren(c);
        for(int i=0;i<children.size();i++){
           if((((Integer)children.elementAt(i)).intValue())==x1){
             //(newnode,xx1)
             this.put("INSERT INTO PCRel VALUES (" + newnode + "," +
                                                    xx1 + ")");
           }else if((((Integer)children.elementAt(i))
                                        .intValue())==x2){
             // (newnode,xx)
             this.put("INSERT INTO PCRel
                                  VALUES("+newnode+","+xx2+")");
           }else{ // (newnode, child)
             this.put("INSERT INTO PCRel VALUES (" + newnode + "," +
                   ((Integer)children.elementAt(i)).intValue() + ")");
           }
        }
```

```
        //Add this specific NXT relationship
        this.put("INSERT INTO NXTRel VALUES("+xx1 + "," + xx2 + ")");
        }
     }
     //Fix NXT Relationships
     Vector nxtrels = getNXTRelationships(this.getNodes(r));
     for(int i=0;i<nxtrels.size();i++){ //For each NXT relationship
        int first =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(0)).intValue());
        int second =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(1)).intValue());
        //If it's not the NXT relationship already added
        if (!(first==n1 && second==n2)){
           if(first == c){ //replace c with newnode
             this.put("INSERT INTO NXTRel VALUES (" + newnode + "," +
                    second + ")");
           }else if (second == c){
             this.put("INSERT INTO NXTRel VALUES (" + first + "," +
                    newnode + ")");
           }
        }
     }
     //Fix RST Relationships
     Vector relations = this.getRSTRelationships(r);
     for(int ii=0;ii<relations.size();ii++){
        int node1 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(0)).intValue();
        int node2 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(1)).intValue();
        int rel_id =
((Integer)((Vector)relations.elementAt(ii)).elementAt(2)).intValue();
        boolean checked = false; //All relations now become unchecked
        if ( !(node1==n1 && node2==n2) ){
           if(c==node1){ //(newnode,node2,rel_id,false)
             this.put("INSERT INTO RSTRel VALUES (" + newnode + "," +
                    node2 + "," + rel_id + "," + checked + ")" );
           }else if (c==node2){ //(node1, new_node, rel_id, false)
             this.put("INSERT INTO RSTRel VALUES (" + node1 + "," +
                    newnode + "," + rel_id + "," + checked + ")" );
           }
        }
     }
     return newnode;
  }
```

(5) The replaceRelationship(…) function has been implemented using the addRelationship() method in the Java. In the design, we used two functions because, in principle, they are two different processes.

(6) The method below corresponds to the removeRelationship(…) function.

```
removeRelationship(n1,n2,r,c){
 1. Store the text of node c in a new node (say, newnode)
 2. Traverse the tree until the specified nodes are found
      (a) if (c!=n1 AND c!=n2)
            x1 = locate (n1, c)
            x2 = locate (n2, c)
            xx1 = removeRelationship(n1,n2,r,x1)
            if (x1 != x2){
                xx2 = removeRelationship(n1,n2,r,x2)
                Fix PC and NXT relationships involving x1 and x2

 3. Fix other relationships
 4. Return ID of newnode
}
```

```
public int removeRelationship (int n1, int n2, int r, int c){
    int newnode = this.newID("Node_ID");
    this.put("INSERT INTO Node VALUES (" + newnode + ",'" +
this.getStringField("Text","Node","ID",c) + "')" );
    int x1 = -10, x2 = -10, xx1 = -10, xx2 = -10;
    if (c!=n1 && c!=n2){
        x1 = this.locate(n1,c); //n1 and n2 have same parent
        x2 = this.locate(n2,c);
        xx1 = this.removeRelationship(n1, n2, r, x1);
        if (x1!=x2){
            xx2 = this.removeRelationship(n1,n2,r,x2);
            //Do not insert any RST relationship
        }
        //Fix PC relationships
        Vector children = this.getChildren(c);
        for(int i=0;i<children.size();i++){
            if((((Integer)children.elementAt(i)).intValue())==x1){
                this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + xx1 + ")");
            }else if((((Integer)children.elementAt(i)).
                                intValue())==x2){
this.put("INSERT INTO PCRel VALUES (" + newnode + "," + xx2 + ")");
            }else{ // (newnode, child)
                this.put("INSERT INTO PCRel VALUES (" + newnode +
"," + ((Integer)children.elementAt(i)).intValue() + ")");
            }
        }
        //Add specific NXT relationship
        if(this.getIntField("Second","NXTRel","First",n1)==n2){
```

```
            this.put("INSERT INTO NXTRel VALUES (" + xx1 + "," +
xx2 + ")");
        }else                                                    if
(this.getIntField("Second","NXTRel","First",n2)==n1){
            this.put("INSERT INTO NXTRel VALUES (" + xx2 + "," +
xx1 + ")");
        }
    }
    //Fix NXT Relationships
    Vector nxtrels = getNXTRelationships(this.getNodes(r)); //Get
all the NXT relationships
    for(int i=0;i<nxtrels.size();i++){ //For each NXT relationship
        int first =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(0)).intValue());
        int second =
 (((Integer)((Vector)nxtrels.elementAt(i)).elementAt(1)).intValue());
        if ((!(first==n1 && second==n2)) &&
                (!(first==n2 && second==n1))){
            if(first == c){ //replace c with newnode
                this.put("INSERT INTO NXTRel VALUES (" + newnode +
"," + second + ")");
            }else if (second == c){
                this.put("INSERT INTO NXTRel VALUES (" + first +
"," + newnode + ")");
            }
        }
    }
    //Fix RST Relationships
    Vector relations = this.getRSTRelationships(r);
    for(int ii=0;ii<relations.size();ii++){
        int node1 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(0)).intValue();
        int node2 =
((Integer)((Vector)relations.elementAt(ii)).elementAt(1)).intValue();
        int rel_id =
 (Integer)((Vector)relations.elementAt(ii)).elementAt(2)).intValue();
        boolean checked = false;
        if ( !(node1==n1 && node2==n2) ){ //Everything except the
relationship to be deleted
            if(c==node1){         //Add         RST         relation
(new_node,node2,rel_id,false)
                this.put("INSERT INTO RSTRel VALUES (" + newnode +
"," + node2 + "," + rel_id + "," + checked + ")" );
            }else   if   (c==node2){   //Add   RST   relation   (node1,
new_node, rel_id, false)
                this.put("INSERT INTO RSTRel VALUES (" + node1 +
"," + newnode + "," + rel_id + "," + checked + ")" );
            }
        }
    }
    return newnode;
}
```

Appendix C

Details of the experiment

C.1 The questionnaire

SECTION I

BACKGROUND INFORMATION

A) Your ID number is: _____

B) Name of your research group (if not DSSE): _____

C) Position:
- ☐ PhD Student
- ☐ Research staff
- ☐ Lecturer
- ☐ Other (please specify): _____

D) On average, how many documents do you produce a month? This includes papers, proposals, lecture notes, presentations, mini-theses and so on.
- ☐ None
- ☐ 1 – 5
- ☐ More than 5

E) How many of these documents are produced jointly with others? _____

F) What was the last document you completed? _____

G) What language do you most often use for communication? _____

H) What technique(s) do you currently use to plan documents? (Tick all that apply)
- ☐ None used
- ☐ Outlines
- ☐ Mind-maps
- ☐ Others (please specify): _____

SECTION II

NARRATIVE ANALYSIS

A) The tutorial/presentation at the start of the experiment was:

- ☐ Good, it covered the concepts well, giving the audience enough information to do the narrative analysis
- ☐ OK, but needs more information on some topics
- ☐ Poor

B) If you answered OK, which sections of the tutorial would you have liked more information on?

_____

C) Did the narrative dictate an appropriate structure for the travel brochure?

- ☐ Yes
- ☐ No

D) If you answered No, which sections of the narrative did you think were particularly poor?

_____

E) How did you find doing the RST analysis of the given document?

- ☐ Very easy
- ☐ Easy
- ☐ Moderate
- ☐ Hard
- ☐ Very hard

F) How long did it take you to do the RST analysis? _____

G) When doing the analysis, did you require more relationship types than those listed in the handout/tutorial?

- ☐ Yes
- ☐ No, the relations in the list were sufficient to do the analysis

H) If you answered Yes to the above, what relations would you have liked to use that were not in the list?

I) Were you able to form a RST tree structure for the narrative?

- ☐ Yes
- ☐ No

J) Did you change any parts of the narrative while doing the analysis to make it fit this tree?

- ☐ Yes
- ☐ No

SECTION III

USING THE TOOL

A) How long did it take you to enter the analysis information into the tool? _____

B) Which of the functions below did you use during the analysis? (Tick all that apply)

- ☐ Read a narrative
- ☐ Create a new narrative
- ☐ Edit text segments
- ☐ Add new text segments
- ☐ Delete text segments
- ☐ Add a subtree to a node in the tree
- ☐ Add relations
- ☐ Remove relations
- ☐ Review relations
- ☐ Merge two versions of the narrative
- ☐ Read a second version of the narrative simultaneously
- ☐ Help

C) The interface will soon be changed to a graphical one. In addition to the functions provided already, do you think any extra functionality is necessary for creating and analysing the narratives collaboratively? If so, please describe them below.

_____

_____

_____

_____

_____

D) Any other comments or suggestions:

_____

_____

_____

SECTION IV

COLLABORATIVE NARRATIVE PRODUCTION

A) During this task, you were in team: A B C (delete as appropriate)

B) How much time did it take to produce the narrative? _____

C) Did you use the tool to create the narrative?
- ☐ Yes
- ☐ No

D) How did you arrive at the final narrative?
- ☐ One member (leader) suggested a narrative and the team revised this
- ☐ Every member contributed sections of the narrative
- ☐ Other (please describe):

_____

E) Did writing a narrative help clarify ideas among the members in your team for the document?
- ☐ Yes
- ☐ No

F) Did you analyse the narrative to see if it was coherent?
- ☐ Yes
- ☐ No

G) Would you consider using a narrative to structure documents in the future?
- ☐ Yes
- ☐ No

Please use this space for any additional comments or e-mail me on nhds03r@ecs.soton.ac.uk. Thank you for taking part in this experiment. Please attach the narrative analysis and other notes before returning this questionnaire.

Thank you very much for your time.

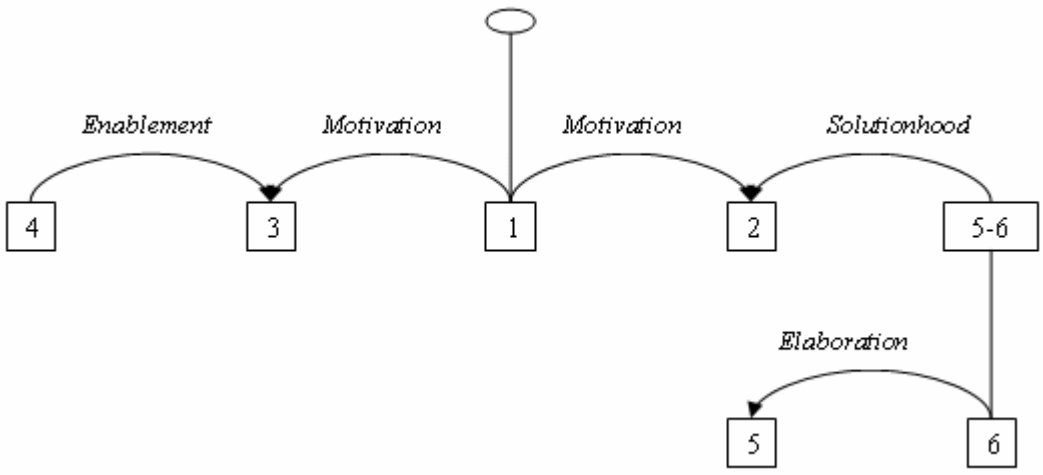
Nishadi De Silva

C.2 RS-trees produced by the volunteers

The DN that the volunteers had to analysis was given in Chapter 8. For each volunteer, we have shown the way he/she segmented the DN and the corresponding RST analysis. Wherever possible, we have used RSTTool to draw the trees.

RST analysis of volunteer 1

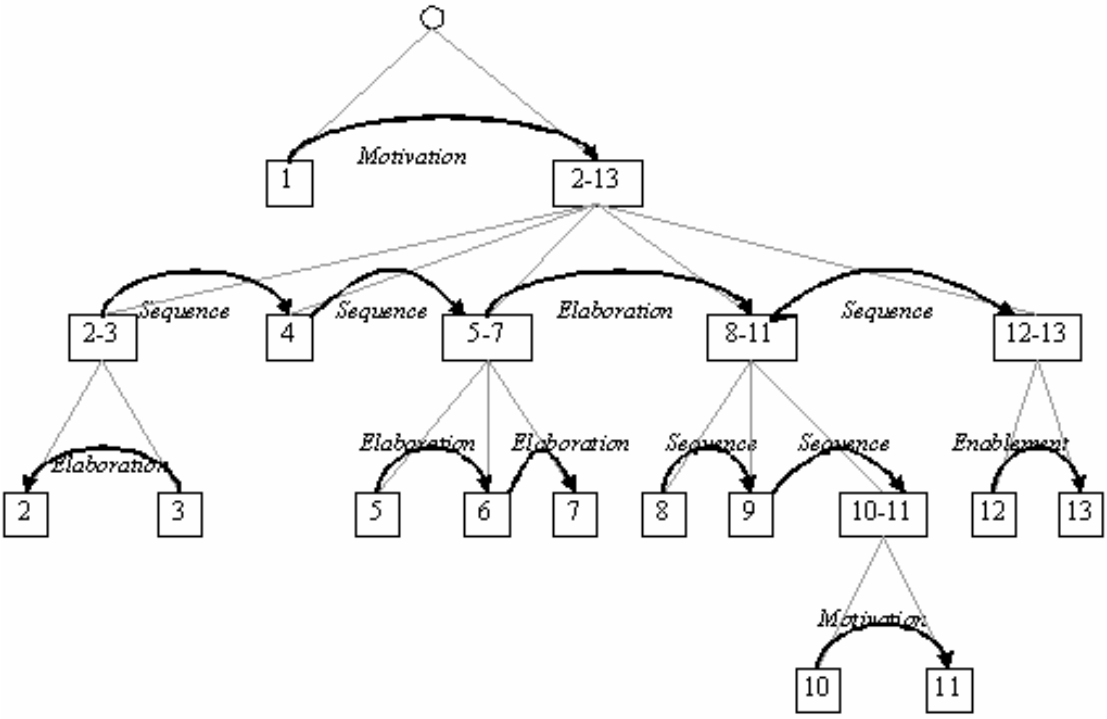
[We want]1 [to convince the reader to book a holiday in the country described.]2 [Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for. The next page will begin with a greeting in the local language and its translation. Five to six short paragraphs will follow this, each describing attractions that will appeal to a wide range of holiday-makers;]5 [some of these attractions will be familiar and some unique so as to distinguish this country from the rest. The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations. Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader (who has read this far).]6 [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers]4 [who may now be considering booking their holidays.]3



The segments in this RST analysis are not in the same sequence as they appear in the DN.

RST analysis of volunteer 2

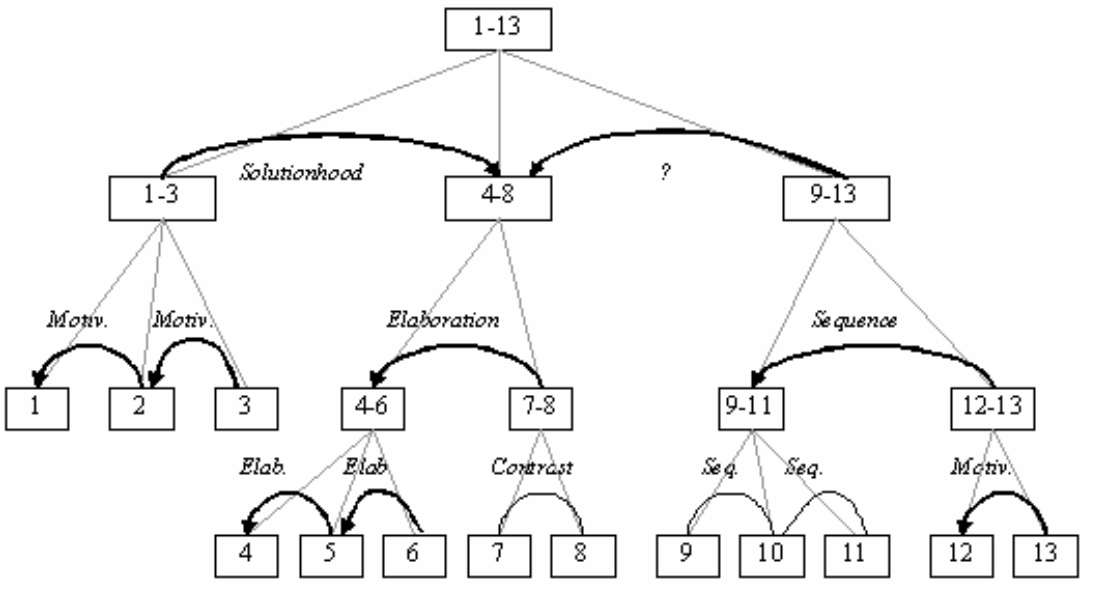
[We want to convince the reader to book a holiday in the country described.][1] [Therefore, on the first page,][2] [we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for.][3] [The next page will begin with a greeting in the local language and its translation.][4] [Five to six short paragraphs will follow this,][5] [each describing attractions that will appeal to a wide range of holiday-makers;][6] [some of these attractions will be familiar and some unique so as to distinguish this country from the rest.][7] [The first of these paragraphs will include a sentence about the country's geographical location][8] [and some of the paragraphs will be enhanced using illustrations.][9] [Next, brief details about the climate, currency and languages spoken will be given][10] [to inform the interested reader (who has read this far).][11] [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers] [12] [who may now be considering booking their holidays.][13]



The RS-tree above is well thought out. The only problem with it is that some of the arrows in the relationships are pointing the wrong way.

RST analysis of volunteer 3

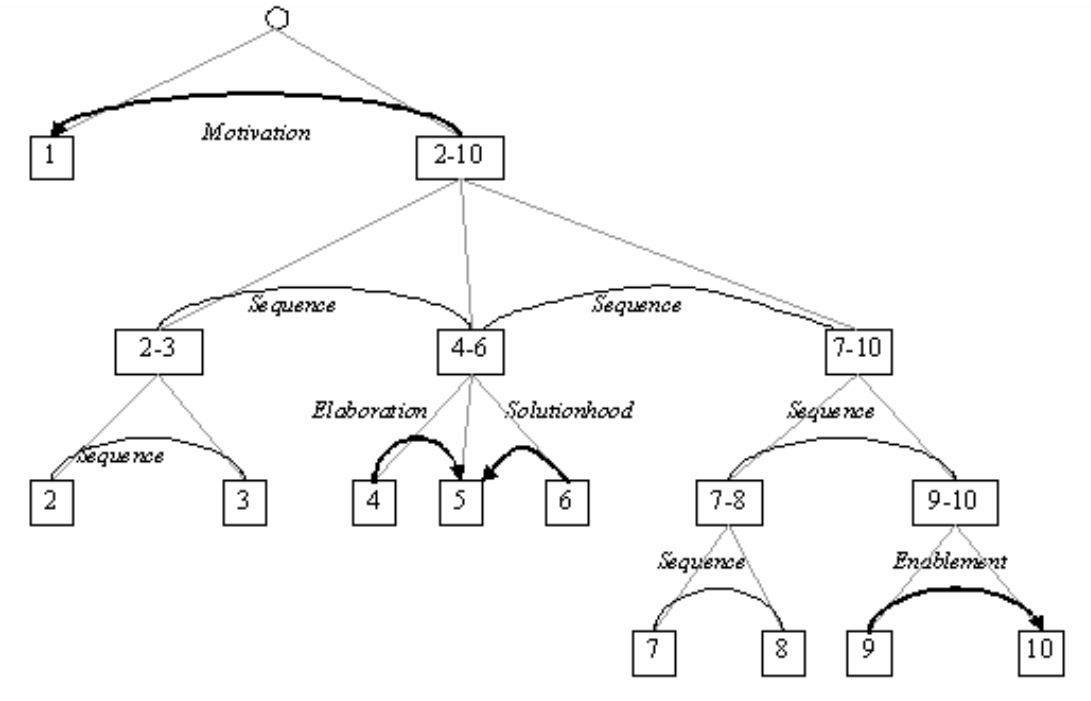
[We want to convince the reader to book a holiday in the country described.][1] [Therefore, on the first page, we'll place a catchy title][2] [and a picture showing a leisurely activity or scenery that this country is famous for.][3] [The next page will begin with a greeting in the local language and its translation.][4] [Five to six short paragraphs will follow this,][5] [each describing attractions that will appeal to a wide range of holiday-makers;][6] [some of these attractions will be familiar][7] [and some unique so as to distinguish this country from the rest.][8] [The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations.][9] [Next, brief details about the climate, currency and languages spoken will be given][10] [to inform the interested reader (who has read this far).][11] [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided][12] [for readers who may now be considering booking their holidays.][13]



Volunteer has put a question mark for the relationship that should link segments 9-13 with segments 4-8.

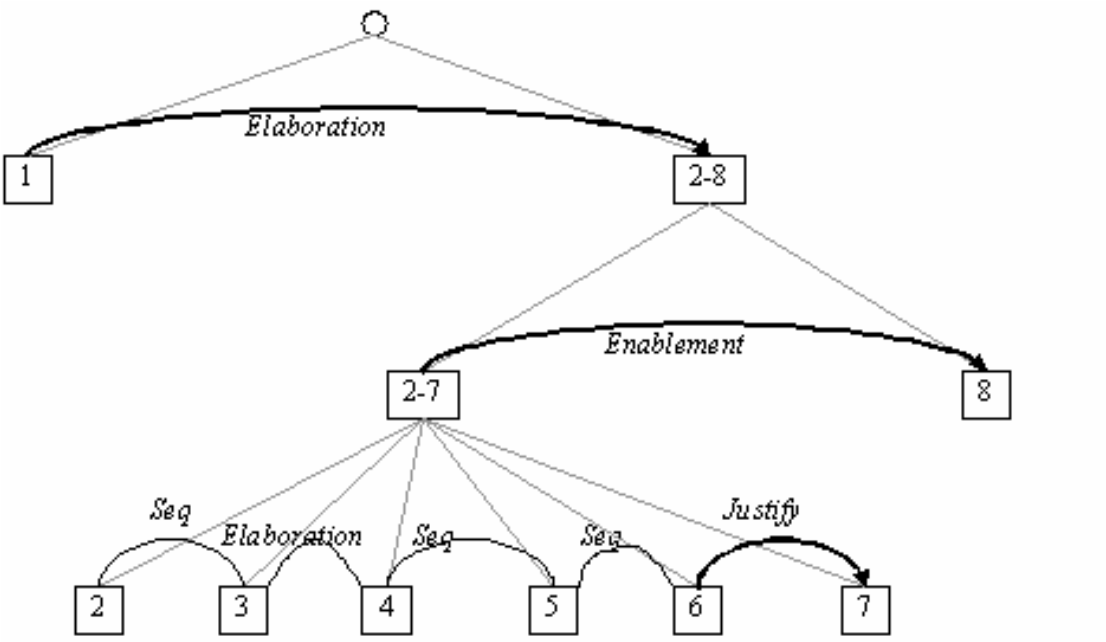
RST analysis of volunteer 4

[We want to convince the reader to book a holiday in the country described.][1] [Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for.][2] [The next page will begin with a greeting in the local language and its translation.][3] [Five to six short paragraphs will follow this, each describing attractions that will appeal to a wide range of holiday-makers;][4] [some of these attractions will be familiar and some unique so as to][5] [distinguish this country from the rest.][6] [The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations.][7] [Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader (who has read this far).][8] [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided][9] [for readers who may now be considering booking their holidays.][10]



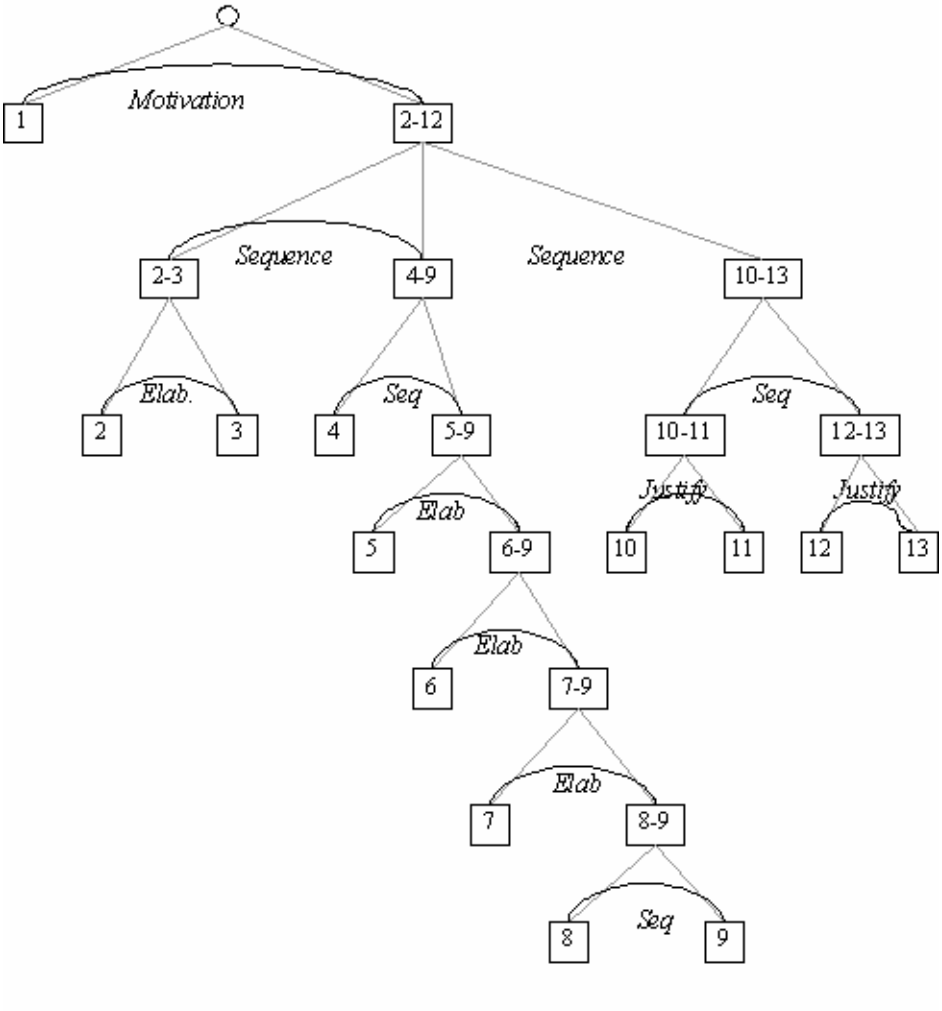
RST analysis of volunteer 5

[We want to convince the reader to book a holiday in the country described.][1] [Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for.][2] [The next page will begin with a greeting in the local language and its translation. Five to six short paragraphs will follow this,][3] [each describing attractions that will appeal to a wide range of holiday-makers; some of these attractions will be familiar and some unique so as to distinguish this country from the rest.][4] [The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations.][5] [Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader][6] [(who has read this far).][7] [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers who may now be considering booking their holidays.][8]



RST analysis of volunteer 6

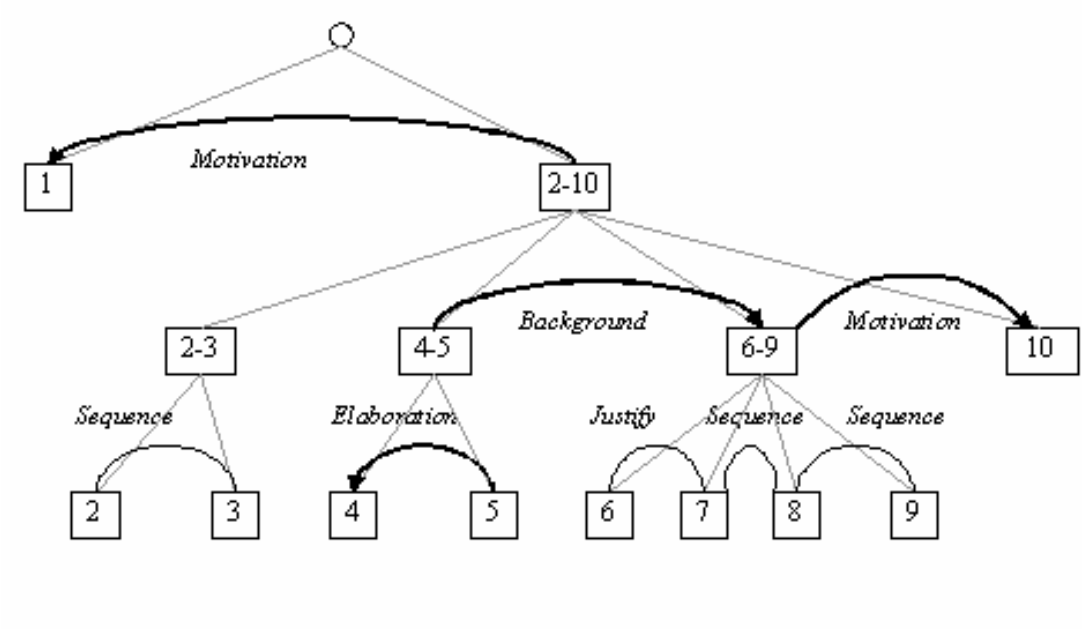
[We want to convince the reader to book a holiday in the country described.][1] [Therefore, on the first page, we'll place a catchy title and a picture][2] [showing a leisurely activity or scenery that this country is famous for.][3] [The next page will begin with a greeting in the local language and its translation.][4] [Five to six short paragraphs will follow this,][5] [each describing attractions that will appeal to a wide range of holiday-makers;][6] [some of these attractions will be familiar and some unique so as to distinguish this country from the rest.][7] [The first of these paragraphs will include a sentence about the country's geographical location][8] [and some of the paragraphs will be enhanced using illustrations.][9] [Next, brief details about the climate, currency and languages spoken will be given][10] [to inform the interested reader (who has read this far).][11] [Finally, contact details of reputable travel agents and a URL][12] [for more information about the country][13] [will be provided for readers who may now be considering booking their holidays.][14]



There appears to be 14 segments in the DN and only 12 in the RS-tree. It is possible that the segmentation was misinterpreted by us because there was a lot of writing (and crossing out) done by this volunteer.

RST analysis of volunteer 7

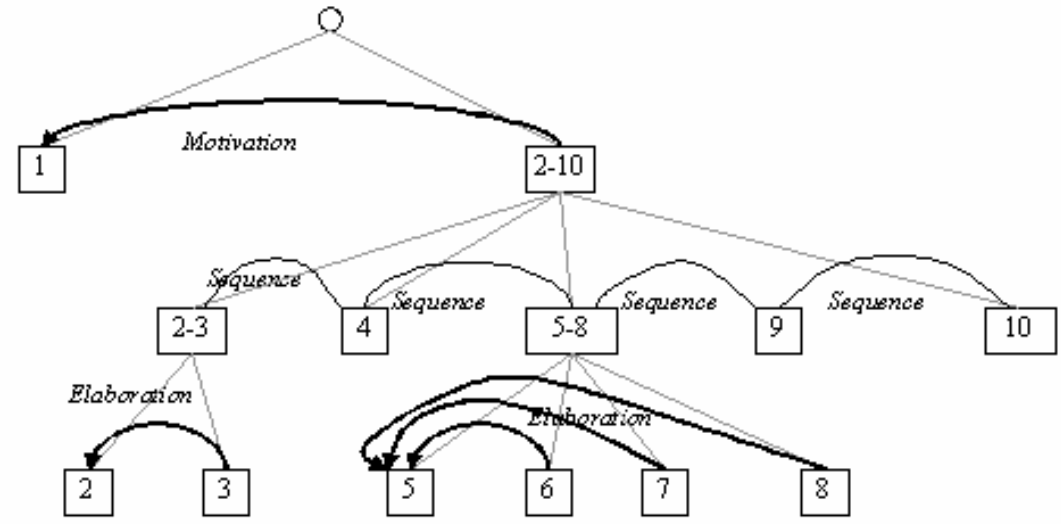
[We want to convince the reader to book a holiday in the country described.]1 [Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for.]2 [The next page will begin with a greeting in the local language and its translation.]3 [Five to six short paragraphs will follow this,]4 [each describing attractions that will appeal to a wide range of holiday-makers;]5 [some of these attractions will be familiar and some unique so as to distinguish this country from the rest.]6 [The first of these paragraphs will include a sentence about the country's geographical location]7 [and some of the paragraphs will be enhanced using illustrations.]8 [Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader (who has read this far).]9 [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers who may now be considering booking their holidays.]10



There was a missing relationship between segment 1 and the span 2-10.

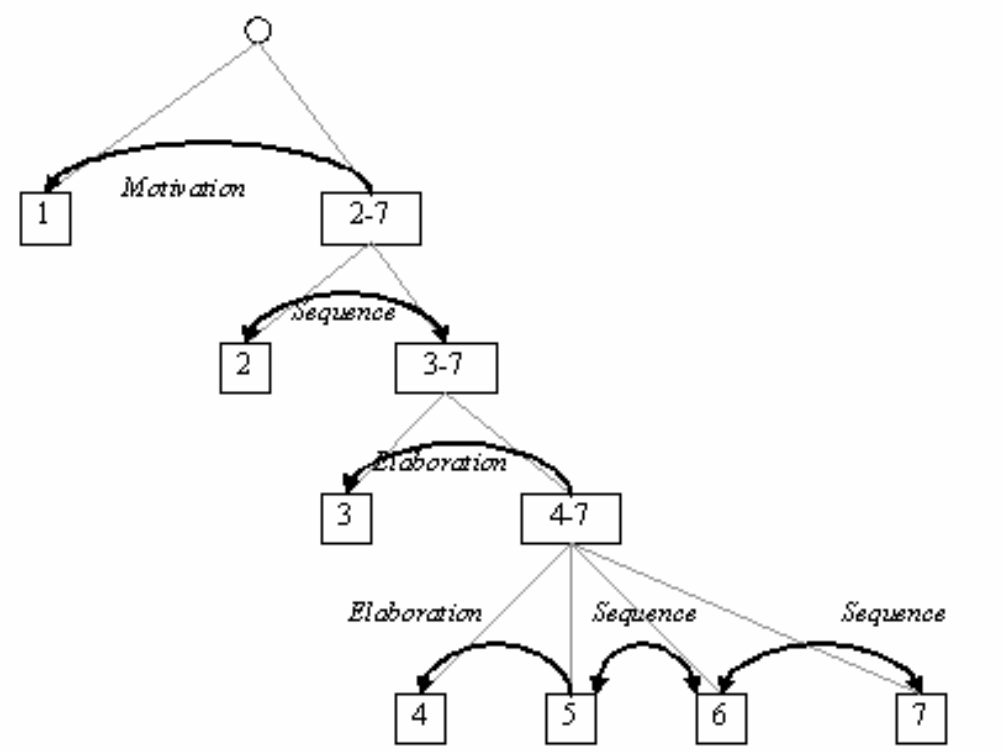
RST analysis of volunteer 8

[We want to convince the reader to book a holiday in the country described.][1] [Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery][2] [that this country is famous for.][3] [The next page will begin with a greeting in the local language and its translation.][4] [Five to six short paragraphs will follow this,][5] [each describing attractions that will appeal to a wide range of holiday-makers;][6] [some of these attractions will be familiar and some unique so as to distinguish this country from the rest.][7] [The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations.][8] [Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader (who has read this far).][9] [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers who may now be considering booking their holidays.][10]



RST analysis of volunteer 9

[We want to convince the reader to book a holiday in the country described.]1 [Therefore, on the first page, we'll place a catchy title and a picture showing a leisurely activity or scenery that this country is famous for.]2 [The next page will begin with a greeting in the local language and its translation.]3 [Five to six short paragraphs will follow this, each describing attractions that will appeal to a wide range of holiday-makers; some of these attractions will be familiar and some unique so as to distinguish this country from the rest.]4 [The first of these paragraphs will include a sentence about the country's geographical location and some of the paragraphs will be enhanced using illustrations.]5 [Next, brief details about the climate, currency and languages spoken will be given to inform the interested reader (who has read this far).]6 [Finally, contact details of reputable travel agents and a URL for more information about the country will be provided for readers who may now be considering booking their holidays.]7



This volunteer had used double-ended arrows for the multi-nuclear relationship SEQUENCE.

List of Figures

Bibliography

AARONSON, J. (2002) Your Web Site As a Narrative Device: Introduction. *CRM Strategies. Article online at http://www.clickz.com/showPage.html?page=1450401 (Last accessed on 23.11.2006).*

ABBOTT, H. P. (2002) *The cambridge introduction to narrative,* Cambridge, UK, Cambridge University Press.

ALRED, G. J., BRUSAW, C. T. & OLIU, W. E. (2003) *Handbook of technical writing,* Boston, MA, Bedford/St. Martin's.

BÄRENFÄNGER, M., HILBERT, M., LOBIN, H. & LÜNGEN, H. (2006) Using OWL ontologies in discourse parsing. *Proceedings of the workshop of Ontologies in Text Technology.* Osnabrück, Germany.

BEAUBOUEF, T. & LANG, R. (1998) Rough Set Techniques for Uncertainty Management in Automated Story Generation. *ACM Southeast Regional Conference 1998.* Marietta, GA, USA.

BERNERS-LEE, T., HENDLER, J. & LASSILA, O. (2001) The Semantic Web. *Scientific American,* 30.

BERNSTEIN, M. (2001) Beyond Usability and Design: The narrative web. *A List Apart (Issue 106). Found online at http://alistapart.com/articles/narrative (Last accessed 15.05.2006).*

BREMOND, C. (1980) The Logic of Narrative Possibilities. IN ONEGA, S. & LANDA, J. A. G. (Eds.) *Narratology.* New York, Pearson Education Inc.

CARLSON, L., MARCU, D. & OKUROWSKI, M. E. (2001) Building a discourse-tagged corpus in the framework of Rhetorical Structure Theory. *2nd SIGdial Workshop on Discourse and Dialogue.* Denmark.

CEDERQVIST, P. (2002) *Version Management with CVS*, Network Theory Ltd.

DE-SILVA, N. & HENDERSON, P. (2005) Narrative Support for Technical Documents: Formalising Rhetorical Structure Theory. *7th International Conference on Enterprise Information Systems (ICEIS).* Miami, FL, USA.

DE-SILVA, N. & SKAF-MOLLI, H. (2006) Narratives to preserve coherence in collaborative writing. *The Eighth International Workshop on Collaborative Editing Systems.* Banff, Canada.

DRIDI, F. & NEUMANN, G. (1999) How to implement Web-Based Groupware Systems Based on WebDAV. *Proceedings, IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.* Stanford, CA, USA.

ECO, U. (1979) *The Role of the Reader: Explorations in the semiotics of texts*, Hutchinson & Co. (Publishers) Ltd.

EULER, J. S. (1992) Against Default Voice: Technical Writing as Speech Act. *IPCC 92 Santa Fe. Crossing Frontiers. Conference Record***,** 125-129.

EVANS, D. & GRUBA, P. (2004) *How to write a better thesis,* Melbourne, Australia, Melbourne University Press.

FELTRIM, V. D. & ALUÍSIO, S. M. (2003) Analysis of the rhetorical structure of computer science abstracts in Portuguese. *Corpus Linguistics.* Lancaster University, UK.

FERBER, J. (1999) *Multi-agent systems: An introduction to distributed Artificial Intelligence*, Pearson Education Limited.

FISH, R. S., KRAUT, R. E. & LELAND, M. D. P. (1988) Quilt: a collaborative tool for cooperative writing. *Proceedings of the ACM SIGOIS and IEEECS TC-OA 1988 conference on Office information systems.* Palo Alto, United States, ACM Press New York.

FOREST, J. (2005) LibreSource: Overview and quick start.

FREYTAG, G. (1863) *Freytag's technique of the drama,* New York & London, Benjamin Blom.

FRIESEN, J. (2004) Java Tech: The ABCs of Synchronization. This is a java.net article available online at http://today.java.net/pub/a/today/2004/08/02/sync1.html (Last accessed 20.9.2006).

GENTHIAL, D. & COURTIN, J. (1992) From Detection/Correction to Computer Aided Writing. *14th conference on Computational linguistics.* Nantes, France.

GOPEN, G. D. & SWAN, J. A. (1990) The science of scientific writing. *American Scientist,* (Nov-Dec 1990), Volume 78**,** 550-558.

GROSZ, B. J. & SIDNER, C. L. (1986) Attentions, Intentions and the Structure of Discourse. *Computational Linguistics*.

GROSZ, B. J., WEINSTEIN, S. & JOSHI, A. K. (1995) Centering: A framework for modeling the Local Coherence of Discourse. *Computational Linguistics,* 21**,** 203-225.

HARRIS, S. & GIBBINS, N. (2003) 3store: Efficient Bulk RDF Storage. *Proceedings of 1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03).* Sanibel Island, Florida.

HENDERSON, P. (2000) Business Processes, Legacy Systems and a Fully Flexible Future. IN HENDERSON, P. (Ed.) *Systems Engineering for Business Process Change.* New York, USA, Springer-Verlag.

HENDERSON, P. & DE-SILVA, N. (2006) A narrative approach to collaborative writing: A business process model. *8th International Conference on Enterprise Information Systems (ICEIS).* Cyprus.

HOBBS, J. R. (1982) Towards an understanding of coherence in discourse. IN LEHNERT, W. & RINGLE, M. (Eds.) *Strategies for Natural Language Processing.* New Jersey, Lawrence Erlbaum Associates, Inc.

HOBBS, J. R. (1985) On the coherence and structure of discourse. Center for the study of language and information, Stanford University.

HUGHES, J. (1989) Why functional programming matters. *Computer Journal,* 32.

JOHNSON, B. (2006) Creator of web warns of fraudsters and cheats. *The Guardian. Available online at http://technology.guardian.co.uk/news/story/0,1938477,00.html (Last accessed 30.11.2006).*

KIERAS, D. E. (1989) An Advanced Computerized Aid for the Writing of Comprehensible Technical Documents. IN BRITTON, B. K. & GLYNN, S. M. (Eds.) *Computer Writing Environments: Theory, Research and Design.* New Jersey, Lawrence Erlbaum Associates.

KNOTT, A. (1996) A Data-Driven Methodology for Motivating a Set of Coherence Relations. *Department of Artificial Intelligence.* Edinburgh, UK, University of Edinburgh.

LANG, R. (1999) A declarative model for simple narratives. *Proceedings, AAAI Fall Symposium on Narrative Intelligence.* North Falmouth, Massachusetts.

LEHNERT, W. (1981) Plot Units: A Narrative Summarization Strategy. IN LEHNERT, W. & RINGLE, M. (Eds.) *Strategies for Natural Language Processing.* New Jersey, Lawrence Erlbaum Associates.

LESTER, J. D. & JAMES D. LESTER, J. (2005) *Writing research papers: a complete guide,* New York, USA, Pearson Education, Inc.

LONG, J. R. (1997) Aesop's fables: Online collection. Found at http://www.pacificnet.net/~johnr/cgi/aesop1.cgi?sel&TheAntandtheGrasshopper&&antgrass.ram (Last accessed: 6.10.2005).

LOTHE, J. (2000) *Narrative in Fiction and Film: An Introduction,* USA, Oxford University Press.

LOWRY, P. B., CURTIS, A. & LOWRY, M. R. (2004) Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice. *Journal of Business Communication,* 41**,** 66-99.

MAHMUD, R. (2004) Revealing Discourse Relations Structure: An Approach for a Dynamic Computer Aided Writing. *Computers and Writing Conference 2004.* Hawaii.

MAHMUD, R. & RAMSAY, A. (2005) Finding Discourse Relations in Student Essays. *Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2005).* Mexico.

MANN, W. & THOMPSON, S. (1988) Rhetorical Structure Theory: Toward a functional theory of text organisation. *Text,* 8**,** 243-281.

MANN, W. C., MATTHIESSEN, C. M. I. M. & THOMPSON, S. A. (1992) Rhetorical Structure Theory and Text Analysis. IN MANN, W. C. & THOMPSON, S. A. (Eds.) *Discourse Description: Diverse Linguistic Analyses of a Fund-Raising text.* Amsterdam and Philadelphia, John Benjamins.

MARCU, D. (2000) *The Theory and Practice of Discourse Parsing and Summarization*, The MIT Press.

MARJANOVIC, O., SKAF-MOLLI, H., MOLLI, P., RABHI, F. & GODART, C. (2006) Supporting complex collaborative learning activities - The LibreSource approach. *8th International Conference on Enterprise Information Systems.* Paphos, Cyprus.

MCCAUGHREAN, G. (1999) *One Thousand and One Arabian Nights*, Oxford University Press.

MINTO, B. (2002) *The pyramid principle,* UK, Pearson Education Limited.

MOLLI, P., OSTER, G., SKAF-MOLLI, H. & IMINE, A. (2003) Using the transformational approach to build a safe and generic data synchronizer. *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work***,** 212-220.

MOORE, J. D. & POLLACK, M. E. (1992) A problem for RST: The need for multi-level discourse analysis. *Computational Linguistics,* 18**,** 537-544.

NEUWIRTH, C. M., CHANDHOK, R., KAUFER, D. S., ERION, P., MORRIS, J. H. & MILLER, D. (1992) Flexible diff-ing in a collaborative writing system. *4th conference on computer-supported cooperative work.* Toronto, Canada, ACM Press.

NEUWIRTH, C. M., KAUFER, D. S., CHANDHOK, R. & MORRIS, J. H. (1990) Issues in the design of computer support for co-authoring and commenting. *3rd conference on computer-supported cooperative work.* LA, California, USA, ACM press.

NEUWIRTH, C. M., KAUFER, D. S., CHANHOK, R. & MORRIS, J. H. (1994) Computer Support for Distributed Collaborative Writing: Defining parameters for interaction. *Conference on computer-supported cooperative work.* Chapel Hill, NC, USA, ACM press.

NOËL, S. & ROBERT, J.-M. (2004) Empirical Study on Collaborative Writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work,* 13.

O'DONNELL, M. (2000) RSTTool 2.4 - A markup tool for Rhetorical Structure Theory. *Proceedings, International Natural Language Generation Conference (INLG'2000).* Mitzpe Ramon, Israel.

ONEGA, S. & LANDA, J. (1996) Introduction. IN ONEGA, S. & LANDA, J. (Eds.) *Narratology.* New York and London, Longman Group Ltd.

ONO, K., SUMITA, K. & MIIKE, S. (1994) Abstract generation based on rhetorical structure extraction. *15th International Conference on Computational Linguistics (COLING'94).* Kyoto, Japan.

PALMQUIST, M. (2003) A brief history of computer support for writing centers and writing-across-the-curriculum programs. *Computers and Composition,* 20.

PARADIS, J. & ZIMMERMAN, M. (2002) *The MIT Guide to Science and Engineering Communication*, The MIT Press.

PARGMAN, T. C. (2003) Collaborating with writing tools: An instrumental perspective on the problem of computer-supported collaborative activities. *Interacting with computers,* 15**,** 737-757.

PORTER, J. (2003) Why technology matters to writing: A cyberwriter's tale. *Computers and Composition,* 20.

PROPP, V. (1928) Morphology of the Folktale. 2nd ed. Austin, University of Texas Press.

REITER, E. & DALE, R. (2000) *Building Natural Language Generation Systems*, Cambridge University Press.

REITTER, D. & STEDE, M. (2003a) Step by step: Underspecified markup in incremental rhetorical analysis. *Proceedings, 4th International Workshop on Linguistically Interpreted Corpora (LINC-03).* Budapest.

REITTER, D. & STEDE, M. (2003b) An underspecified markup syntax for rhetorical structure annotations.

RIZZO, P., SHAW, E. & JOHNSON, W. L. (2002) An agent that helps children author rhetorically-structured digital puppet presentations. *Proceedings, 6th International Conference on Intelligent Tutoring Systems.*

RÖSNER, D. & STEDE, M. (1992) Customising RST for the Automatic Production of Technical Manuals. IN DALE, R., HOVY, E., RÖSNER, D. & STOCK, O. (Eds.) *Aspects of Automated Language Generation.* Berlin, Springer.

ROTH, A. J. (1999) *The research paper: process, form and content,* USA, Thomas Learning Inc.

SHARPLES, M. (1996) An Account of Writing as Creative Design. IN LEVY, C. M. & RANSDELL, S. (Eds.) *The science of writing: Theories, Methods, Individual Differences and Applications.* New Jersey, Lawrence Erlbaum Associates.

SOWA, J. F. (1983) *Conceptual Structures: Information processing in mind and machine,* New York, Addison-Wesley.

STEELE, V. (2002) Using mind maps to develop writing. *Articles on teaching English by the BBC and British Council. Available online at: http://www.teachingenglish.org.uk/think/write/mind_map.shtml (Last accessed on 1.8.06).*

TABOADA, M. & MANN, W. C. (2006a) Applications of Rhetorical Structure Theory. *Discourse Studies,* 8.

TABOADA, M. & MANN, W. C. (2006b) Rhetorical Structure Theory: Looking back and moving ahead. *Discourse Studies,* 8.

TICHY, W. F. (1982) Design, implementation, and evaluation of a Revision Control System. *6th international conference on Software engineering.* Tokyo, Japan, IEEE Computer Society Press.

TICHY, W. F. (1985) RCS: A System for Version Control. *Software: Practice & Experience,* 15**,** 637-654.

TORRANCE, M. & BOUAYAD-AGHA, N. (2001) Rhetorical Structure Analysis as a method for understanding writing processes. IN DEGAN, L., BESTGEN, Y., SPOOREN, W. & WAES, L. V. (Eds.) *Multidisciplinary Approaches to Discourse.* Amsterdam, Nodus.

TUFFIELD, M. M., MILLARD, D. E. & SHADBOLT, N. R. (2006) Ontological Approaches to Modelling Narratives. *2nd AKT DTA Symposium, AKT.* Aberdeen University, UK.

WHEELER, K. (2004) Freytag's pyramid. Available online at http://web.cn.edu/kwheeler/documents/Freytag.pdf (Last accessed on 26.10.2006).

WINOGRAD, T. (1999) Documentation, Interaction, and Conversation. *The Journal of Computer Documentation,* 23**,** 3-6.

XINDICE (2004) Available online at http://xml.apache.org/xindice/ (Last accessed on 25.6.2004).

ZOBEL, J. (2004) *Writing for computer science,* USA, Springer.