EU-IST Project IST-2003-506826 SEKT
SEKT: Semantically Enabled Knowledge Technologies

D2.3.2 Deep Structure Analysis V2

James Allen (BT)

Abstract
This document accompanies the D2.3.2 software deliverable and contains two major sections. The first describes the architecture, design and usage of the D2.3.2 deliverable. This deliverable annotates a document with the document's rhetorical structure. The annotation is done using a .NET web service and can be accessed directly or using a wrapper that allows it to be used as a standard GATE module. Instructions for incorporating this module in larger software projects are included for both methods.

The second part of the document describes in detail the theory behind the module including an overview of Rhetorical Structure Theory (the framework used for the analysis) and a description of the procedure for generating the rules used to make the annotations. It then continues with a description of possible future research directions.

Those only interested in using the module as a black box can of course ignore section two while those interested in the theory used but not intending to implement a system using it can concentrate on section 2. Either section can be read in isolation.

Keyword list: RST, Rhetorical Structure Theory, Deep Structure Analysis, automatic learning.

WP2 Metadata Generation
Report
31st December 2005

PU
30th January 2006

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540
Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592
Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891
Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

University of Innsbruck
Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475
Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006
Madrid
Spain
Tel: +34 913 349 797
Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00
Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912
Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Sirma Group Corp., Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vall` es)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Siemens Business Services GmbH & Co. OHG
Otto-Hahn-Ring 6
81739 Munich
Germany
Contact person: Dirk Ramhorst
Tel: +49 (89)63640225; Fax: +49 89 63640233
Email: Dirk.Ramhorst@siemens.com

Executive Summary

This document is in two parts. The first is a guide to the use of an Rhetorical Structure Theory (RST) module developed by BT for the SEKT project. The second section discusses the research and theory behind the module's implementation and outlines possible future research directions.

The basis for this module, RST, is a method of determining the overall structure of a document via the use of surface features. These features could include the position of the text within the document, keywords in the text or changes in vocabulary between sections of the text. While RST will say nothing about the meaning of the text it can indicate the structural relationship between parts of the text.

For example in this Executive summary:

- Paragraph 1 is an introduction
- Paragraph 2 elaborates on the subject introduced in paragraph 1
- Paragraph 3 is a shift in topic from paragraph 2 and again elaborates the initial paragraph.

The information gained from an RST parse is usually provided in a tree structure with nodes being labelled with the relation connecting the branches (usually 2 of them) coming from it. Individual branches are labelled with either "nucleus" or "satellite" depending on whether they are considered to be of primary or secondary importance in their particular relation. This information can be directly useful in the summarization of documents or provision of navigation maps for large documents or documents displayed on small screen devices. Additionally, as only surface features are used, RST can potentially analyse documents in real-time, currently circa 0.7 seconds for a newspaper article, and provide this information to other systems that do deeper processing of the document, such as question answering systems or, pertinently to SEKT, automatic ontology generation.

The first two years of the SEKT project implemented the general RST parsing approach described above. This system used a corpus of human annotated documents to train a classifier that could then analyse arbitrary text and add annotations. These annotations could then be used to construct an RST tree for the text. This system was provided both as a standalone .dll or .NET web service that could take text or html and return a document with appropriate mark-up. Additionally, to aid users of Sheffield University's GATE language processing software, a java stub was provided that could take an arbitrary GATE document, reparse it, call a web service and add a new set of annotations to the original GATE document. The modified GATE document could then be viewed or used by any other GATE module. The annotations added to the GATE document also contained sufficient information to fully reconstruct the RST tree.

The feasibility of this procedure and an initial, simulated, annotation were the focus of the first year's work. This working automated system and this document [D2.3.2] were the output of this year's research. The framework built in year one was extended to be more of a testing platform and was then used to collect data from a corpus, train a machine learning algorithm from that data and use this algorithm to annotate arbitrary documents. The architecture of the system is very open and

designed to allow rapid modifications to both the machine learning and parsing sections allowing different RST parsing algorithms to be implemented relatively simply.

While the framework has been a success, the first annotation attempts have led to the conclusion that insufficient data exists for the techniques used to function at an acceptable or useful level. In general**,** it is possible that very large amounts of training data with a reduced number of relations would deliver improved results. However, this data (i) is hard to create for real-life applications since it requires skilled annotators (ii) requires significant effort to create and (iii) such an activity is tangential to SEKT's main focus. It is recommended that further investigation along these lines is suspended.

Contents

SEKT Consortium **2**
Executive Summary **3**
Contents **5**
1 Architecture, Design and Use of D2.3.2 ..... **6**
1.1 Introduction 6
1.1.1 Brief introduction to RST 6
1.1.2 Summary of Module Purpose 7
1.2 System Design and Architecture 8
1.2.1 Access Methods 8
1.2.2 Annotation Mechanism 9
1.3 System User Guide – Standalone 9
1.3.1 Input 9
1.3.2 Output 10
1.4 System User Guide – Web Service 11
1.4.1 Input 11
1.4.2 Output 12
1.5 System User Guide – GATE module 13
1.5.1 Input 13
1.5.2 Output 13
1.5.3 Java Call Code 14
2 Module Theory and Research Directions **16**
2.1 Introduction 16
2.1.1 Statistical Processing 16
2.1.2 Surface level document analysis 16
2.1.3 Rhetorical Structure Theory 16
2.2 Outline of Rhetorical Structure Theory 17
2.3 Clause Finding 18
2.4 Relation assignations (Cue Phrases) 18
2.5 Tree Building 19
2.6 Rule Learning Algorithm 19
2.6.1 Training 19
2.6.2 Annotating 20
2.7 Current Module Performance 22
2.7.1 Evaluation Method 22
2.7.2 Results 23
2.8 Research Directions 23
2.9 Conclusions 24
3 Bibliography and references **25**
4 Index **26**

1 Architecture, Design and Use of D2.3.2

1.1 Introduction

This document forms part of the deliverable D2.3.2 Deep Structure Analysis[1] for the SEKT project. In this document the software component of the deliverable is described and a user guide provided along with a description of the research completed in order to develop this module.

1.1.1 Brief introduction to RST

Rhetorical Structure Theory (RST) [5] is one of a number of theories on "discourse structure" an area of language research concerned with analysing the structure of documents rather than the contents. The general idea of discourse structure is to say how various sections (clauses, sentences, paragraphs) of the document relate to one another without saying what those sections are about. For example finding justifications or explanations for statements, finding sections that show contrasting opinions or argue that a viewpoint is flawed.

RST, unlike some other discourse theories, attempts to analyse this structure without doing any **deep parsing**. It uses **surface features** of the document only. These surface features could include punctuation, particular **cue phrases** or the distribution of words throughout a document. It does not make any attempt to understand the meaning of the document or relationships between words.

A typical RST parse would split the document into **clauses** using the punctuation of the document, then each clause would become a leaf in a tree structure. Every node in the tree up to (and including) the root node would be labelled with a **relation**. These relations could be "justification", "topic change", "elaboration" or one of many others. The exact set of relations used depends upon the choice of the implementer.
Each branch of the tree would be labelled either as a **nucleus** or **satellite** depending on whether it was of primary or secondary importance in the relation. For example in a "list" relation all branches could be nucleus, i.e. of equal importance, whereas in an elaboration the main section would be the nucleus and the section elaborating on it would be the satellite.

The end result of an RST parse should be a tree that at its lowest levels shows the relationships between individual clauses, at higher levels the relationships between sentences and paragraphs. In longer documents it could even show the relationships between chapters at the highest levels. Figure 1 shows an example of a small RST tree.

[1] This has been seen as confusing as the task is Deep Structure Analysis but only surface cues are used. It is meant to indicate that the Deep Structure of the document is found by using these "shallow" parsing techniques.

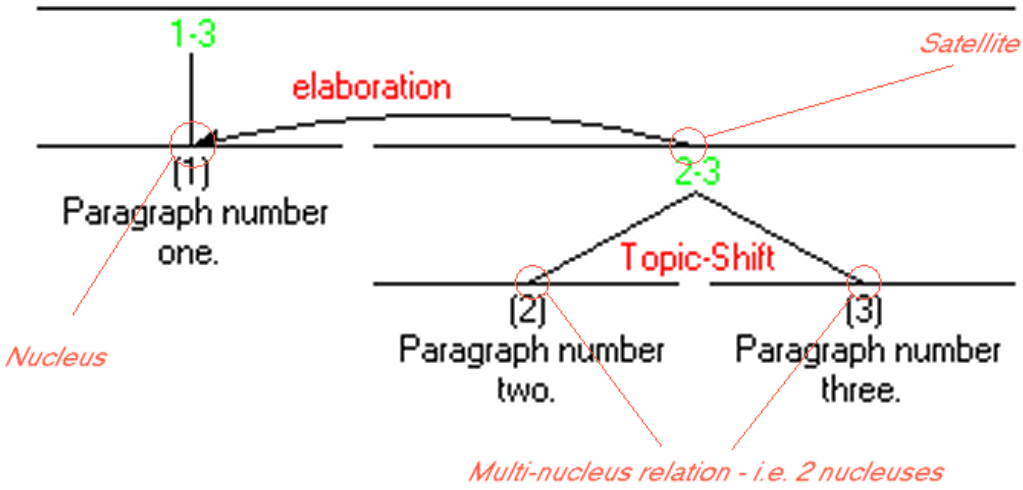

Figure 1 small RST tree

For a detailed and in depth look at RST and discourse analysis in general please see section 2, "**Module Theory and Research Directions**."

1.1.2 Summary of Module Purpose

The D2.3.2 software module is intended to be a "black box" RST annotator that will take a piece of text and return an RST tree for that text. This black box can be accessed either as a .NET web service, a GATE module or accessed directly as a .dll. The multiple forms of access allows our colleagues at the University of Sheffield, and anyone else who uses their GATE document handling software, to easily integrate the software module in a Linux based environment using Java code. At the same time it allows the module to be integrated with BT's Windows based C# code.

In its basic form the module takes a document in text or html format as input and returns a set of annotations. These annotations label either leaves or nodes. The leaves span a piece of text and contain the label for that text (a number between 1 and N) and the id of the text's parent. The nodes contain the id of the left and right branches, the node's parent node id, the relation between the branches and whether the branches are nuclei or satellites. From this information the RST tree structure is built up in whatever format best suits the objectives of the module user.

1.2 System Design and Architecture

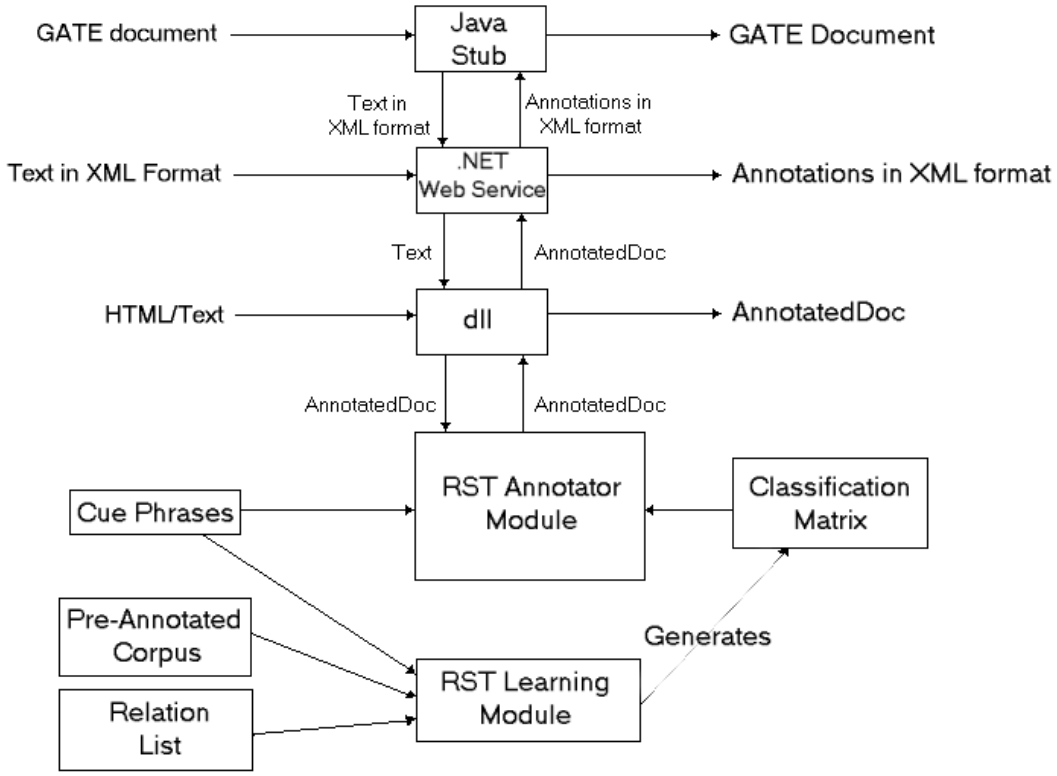

Figure 2 Module design architecture

The architecture shown above in Figure 2 is built around some of BT's existing language processing software which performs a similar function to Sheffield's GATE system. The software will take in "documents" which can be in text, html or XML format. It can then add sets of annotations to those documents. The combination of document and annotations is known as the "AnnotatedDoc" format. Each set of annotations consists of an arbitrary number of annotations and a set name. Each annotation has a left and right pointer which point to the annotation's span in the text of the document. The annotations also have a set of user defined fields and values.

1.2.1 Access Methods

Once a document has been annotated with a set of RST relations it can then be supplied either directly as an AnnotatedDoc or as a modified version of the original text/html. Usually if the module is being accessed directly from user code or via the .dll it is more useful to use the AnnotatedDoc representation. There is also an option to output a .dis file which contains the RST mark-up and text in a machine/human readable format.

Additionally a .Net web service can be used as an alternative interface. This takes in the text of a document in XML format, reads it into AnnotatedDoc format, calls the RST annotator and returns an XML list of the RST annotations that can be applied to the original document to construct the RST tree. This web service can be called on the local machine or remotely from any operating system that supports web service calls.

Finally, a Java stub is provided that will take in a GATE document and call the web service described above with the text from that document. The stub can then take the returned XML list and add the annotations from it to the GATE document.

1.2.2 Annotation Mechanism

The underlying annotation mechanism for all three access methods is the same. The text is split into clauses and each clause is assigned an ID. The RST annotator takes in a list of cue phrases and scans each clause for these phrases, noting any matches. It then compares matches with a matrix which records the number of occurrences of cue phrases and the relations they indicate. Additional information indicates whether the fragment containing the cue phrase was a nucleus or satellite and which level of the tree the relations appeared at. This information is used to select the most likely relation, select the nucleus/satellite assignments and choose at which level in the tree the relation should appear. For a more detailed explanation of this annotation process see 2.4 Relation assignations (Cue Phrases) onwards. Once these annotations are in place they can then be supplied to other modules.

The classification matrix is compiled offline using a training corpus of pre-annotated texts. Lists of cue-phrases and relations are imported. The corpus is then examined and every co-occurrence of a relation and a cue phrase is stored along with data about where in the tree it was found and what the nucleus/satellite values were. These figures are then weighted (based on frequency of the cue-phrase and relation in question) and used to create the matrix. Again for more detail on this process please see 2.4 Relation assignations (Cue Phrases).

1.3 System User Guide – Standalone

The module can be provided as a .dll and accessed by any standard COM methods.
The following gives the processes for using the modules as part of a C# .NET module.

1.3.1 Input

The standard method for annotating a document in C# would be:

```
using System;
using RSTRuleUser;
using LTG;
using LTG.AnnotatedDocReaders;
using LTG.Annotators;
using System.IO;
public class Tester {
  public Tester() {}
  public static void Main(string[] ARGS) {
      string MatrixName=@"c:/GATE/RSTCorpus/fullMatrix.txt";
      string CuePhraseName=@"c:/GATE/RSTCorpus/cuePhrases.txt";
      string RelationName=@"c:/GATE/RSTCorpus/relations.txt";
      string InputName="wsj_0623.out.edus";

      // Loads up the matrix and prepares reusable objects.
      RSTAnnotator ra=new RSTAnnotator();
      ra.Initialise(MatrixName,CuePhraseName,RelationName);
      AbbrvAnnotator aa=new AbbrvAnnotator();
```

```
        SentenceAnnotator sa=new SentenceAnnotator();
        // Load in text file
        ReadTXT rt=new LTG.AnnotatedDocReaders.ReadTXT();
        FileStream fr=new FileStream(InputName,FileMode.Open);
        AnnotatedDoc ad=rt.ImportDoc(fr);
        ad.StoreSpans=true;

        // Split document into sentences.
        aa.annotate(ref ad);
        sa.annotate(ref ad);

        // Annotate with RST
        ra.Annotate(ref ad,"Sentence","RST");
    }
```

The document would then be contained in the AnnotatedDoc ad with a set of annotations that could be accessed.

1.3.2 Output

There are several options for getting output.
To just dump the annotations to screen use:

```
Console.WriteLine(ad.getAnnotationSet("RST").ToString());
```

To get a .dis file output use:

```
StreamWriter sw=new StreamWriter(@"C:\outfile.dis");
sw.Write(pd.output2Dis(ref ad,"RST"));
sw.Close();
```

To export an XML file which only contains the annotations use:

```
ReadGATE rg=new ReadGATE();
rg.ExportDoc(@"C:\output.xml",ref ad);
```

Individual annotations can be accessed and particular fields can be read from them:

```
AnnotationSet aset=ad.getAnnotationSet("RST");
foreach(Annotation a in aset) {
      Console.WriteLine("({0},{1},{2})",
                        a.Left,a.Right,ad.getText(a));
      Console.WriteLine("ID={0}",a.Attributes["NodeID"]);
}
```

The attributes available for each annotation are as follows:

Field Name	Value
NodeID	A number between 1 and N that is the unique identifier for that node in the tree.
Nuclear_Type	The nuclear type of the node. Either "Nucleus" or "Satellite"
Relation	The relation between the 2 branches of that node. This value can be null if the node in question is a leaf of the tree.
Parent	The Node ID of the node's parent. Will be -1 for the root Node.
Left_Child	The NodeID of the node's left hand branch. Will be -1 if the node is a leaf node.
Right_Child	The NodeID of the node's right hand branch. Will be -1 if the node is a leaf node.
Text_Only	Will be "-1" unless the node is a leaf node. Leaf nodes will be numbered 1,2.. ,N from left to right in this field.

Table 1 Available fields in individual RST annotations.

And of course with the annotations and document stored in memory, custom output routines can be written – these are however beyond the scope of this document.

1.4 System User Guide – Web Service

1.4.1 Input

A second method for using the RST annotator is via the RSTWebService. The RSTWebService can be accessed using any standard web service method. How exactly this is done will depend upon your choice of environment and programming language. When the web service is accessed you will be able to use the procedure RSTAnnotate. This procedure takes as input a single string. This should be the text to be annotated formatted as XML as follows:

```
<?xml version="1.0" encoding="utf-16"?>
<Data>The text to be annotated.</Data>
```

Figure 1 Example input to Web Service

Additional XML tags can be added, should you wish, such as DTD information, a generic <XMLDocumentType> tag or any other tags (outside the Data section). Everything but the contents of <Data> </Data> will be ignored however. This allows users to use more generic multipurpose XML formats.

1.4.2 Output

The RSTAnnotate procedure will return only the annotations (not the text) in the form of an XML document. These annotations can be used to reconstruct the generated binary branching RST tree[2].

```
<?xml version="1.0" encoding="utf-16"?>

<Anno2GateDoc>
<RST>
      <RSTTag NodeID="1" Left="4" Right="90" Nuclear_Type="Satellite"
      Relation="elaboration-additional" Left_Child="-1"
      Right_Child="-1" Parent="5" Text_Only="1" />
      <RSTTag NodeID="5" Left="4" Right="181" Nuclear_Type=""
      Relation="Continuation" Left_Child="1" Right_Child="4"
      Parent="-1" Text_Only="-1" />
      <RSTTag NodeID="2" Left="94" Right="129"
      Nuclear_Type="Satellite" Relation="elaboration-additional"
      Left_Child="-1" Right_Child="-1" Parent="4" Text_Only="2" />
      <RSTTag NodeID="4" Left="94" Right="181" Nuclear_Type="Nucleus"
      Relation="elaboration-additional" Left_Child="2"
      Right_Child="3" Parent="5" Text_Only="-1" />
      <RSTTag NodeID="3" Left="133" Right="181"
      Nuclear_Type="Nucleus" Relation="elaboration-additional"
      Left_Child="-1" Right_Child="-1" Parent="4" Text_Only="3" />
</RST>
</Anno2GateDoc>
```

Figure 4 Example output from Web Service

[2] Note that unlike the trees generated by Marcu (among others) the trees from the web service are binary branching. So where 3 or more list elements would have branched from a single node they will now be split into binary branches. This was a delibarate decision as it was felt that strictly binary trees would be easier to process.

For each annotation the fields have the following meaning:

Field Name	Value
NodeID	A number between 1 and N that is the unique identifier for that node in the tree.
Left	A number between 0 and DocLength which is the leftmost bound of the text spanned by the node and it's branches.
Right	A number between 0 and DocLength which is the rightmost bound of the text spanned by the node and it's branches.
Nuclear_Type	The nuclear type of the node. Either "Nucleus" or "Satellite"
Relation	The relation between the 2 branches of that node. This value can be null if the node in question is a leaf of the tree.
Parent	The Node ID of the node's parent. Will be -1 for the root Node.
Left_Child	The NodeID of the node's left hand branch. Will be -1 if the node is a leaf node.
Right_Child	The NodeID of the node's right hand branch. Will be -1 if the node is a leaf node.
Text_Only	Will be "-1" unless the node is a leaf node. Leaf nodes will be numbered 1,2.. ,N from left to right in this field.

Table 2 Output fields from web service.

It is worth noting that the Left and Right fields are numbered according to the gaps between characters. The point before the first character in the document is point 0 and the point between the first and second characters is point 1. The point directly after the last character (assuming the document has N characters) will be point N.

In addition the Left and Right markers will mark portions of text, leading or trailing spaces will not be included. This behaviour seems most useful as it will depend upon the user's purpose whether white space should be ignored or treated as part of the preceding or following fragment.

1.5 System User Guide – GATE module

1.5.1 Input

This module is intended to be used as a standard GATE processing resource. As such it should be loaded within the GATE framework, put into a pipeline and passed any GATE document containing at least some text.

1.5.2 Output

The result of this module will be the original GATE document with an additional annotation set (defaulting to the name RST) this additional set will contain the features with the same values and names as shown in 1.4.2.

1.5.3 Java Call Code

The java code is used to allow GATE systems to pass in a GATE document, call the web service described in 1.4 above, and add the resulting annotations to the GATE document. It is anticipated that users may wish to alter this code to reflect either changes to the location of the web service (such as running a local copy to improve processing speed) or for use in a more specialised GATE application. As such some detail of the code structure is given below.

The Java code currently consists of 7 classes. Four of these classes are automatically generated by the axis 1.1[4] WSDL2Java utility. These four[3] are generated to access a specific web service. The remaining 3 were coded specifically for the SEKT project and consist of a main class – genAnn.java and two less important helper classes that are there mainly to keep the code tidy.

Changing the web service hostname

Hopefully this code should be fairly static. The main exception to this rule is that should the web service change host the following changes must be made.
In Service1Locator.java:

```
private final java.lang.String
      Service1Soap_address = "http://oldhost/RSTWebService/RST.asmx";
```

and

```
return new javax.xml.namespace.QName(
      "http://oldhost/RSTWebService/", "Service1");
```

should become

```
private final java.lang.String
      Service1Soap_address = "http://newhost/RSTWebService/RST.asmx";
```

and

```
return new javax.xml.namespace.QName(
      "http://newhost/RSTWebService/", "Service1");
```

And the following changes to Service1SoapStub.java:

```
oper.addParameter(new javax.xml.namespace.QName(
      "http://oldhost/RSTWebService/", "xmlInput"....
oper.setReturnQName(new javax.xml.namespace.QName(
      "http://oldhost/RSTWebService/", "RSTAnnotateResult"));
_call.setSOAPActionURI("http://oldhost/RSTWebService/RSTAnnotate");
_call.setOperationName(new javax.xml.namespace.QName(
"http://oldhost/RSTWebService/", "RSTAnnotate"));
```

Should be changed, respectively, to:

```
oper.addParameter(new javax.xml.namespace.QName(
      "http://newhost/RSTWebService/", "xmlInput"....
oper.setReturnQName(new javax.xml.namespace.QName(
      "http://newhost/RSTWebService/", "RSTAnnotateResult"));
_call.setSOAPActionURI("http://newhost/RSTWebService/RSTAnnotate");
_call.setOperationName(new javax.xml.namespace.QName(
"http://newhost/RSTWebService/", "RSTAnnotate"));
```

Of course this can be done most easily by doing a search and replace on “oldhost”.

[3] Service1.java, Service1Locator.java, Service1SoapStub.java and Service1Soap.java where Service1 is the name of the service being accessed.

The rest of these files should not need to be altered while the web service interface remains constant.

Other Changes

The file genAnn.java contains the GATE portion of the code. If a specific behaviour is required, e.g. some form of pre-parsing then this is the file that should be altered.

Required Libraries

This code uses the gate.jar library [3] and the axis 1.1 libraries[4] [4] both of which can be downloaded from the internet and have extensive documentation on the download sites.

The screenshot in figure 5 shows some results from the RST module within GATE.

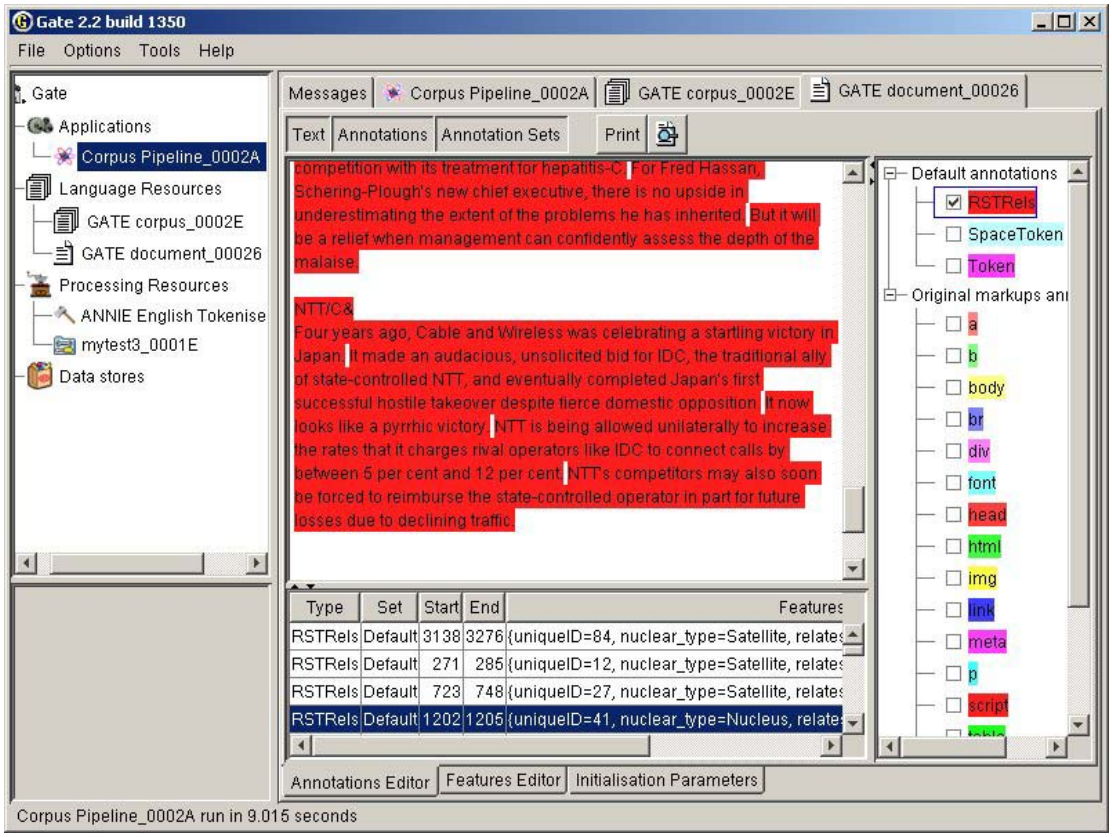

Figure 5 The RST module within GATE

[4] They are axis.jar, axis-ant.jar, common-discovery.jar, common-logging.jar, jaxrpc.jar, log4j-1.2.8.jar, saaj.jar and wsdl4j.jar

2 Module Theory and Research Directions

2.1 Introduction

2.1.1 Statistical Processing

There are two major approaches to language processing problems:
The first is the "linguistically motivated" approach. This uses huge knowledge bases, vast rule sets, deductively to generate logical and rational methods of extracting meaning from a given text. It often does not work. It also takes a lot of people long time to do and is completely language dependent. Changing language involves starting again from scratch.

The second involves using training corpora to provide data for various statistical methods of machine learning. This has the advantage that the core software can remain the same and simply using new training data will prepare the system for using a new language. As the SEKT project is split across several languages this potential language independence was valuable. Although as of this time the only known training set existing is in English, creating training data requires semi-skilled annotators and existing software. Creating language processing software requires highly skilled linguists and computer scientists.

2.1.2 Surface level document analysis

Surface level analysis involves using only the characters that make up a text to derive some sort of information. As opposed to deep analysis which could involve transforming the characters into words and those words into related concepts and using these concepts to generate meaning from the text. For example surface level sentence splitting could involve finding a "." character followed by a new-line or space where the next word started with a capital letter. The deep analysis would involve understanding the meaning of the text and using its meaning to detect changes in topic that indicated likely sentence breaks.

It was decided, in the case of this deliverable, to concentrate on the surface level features of the documents to be processed. This has the advantages of being faster, computationally (deep analysis is very hard to do in real-time), easier to implement and less language dependent.

2.1.3 Rhetorical Structure Theory

There are several theories proposing ways in which documents may be organized or structured. This deliverable concentrated on Rhetorical Structure Theory and was based on the work by Marcu [2]. This implementation of the theory uses surface cues to break the document up into sections and relate those sections to each other. As, unlike other discourse theories, it relies only on surface cues it can process documents in real time[5] and does not need in depth linguistic knowledge for its use or

[5] For this system processing took around 0.7 seconds on a standard desktop PC for a typical newspaper article.

implementation. The end product is a document arranged in a tree structure with each clause or sentence as a leaf node and the relations marked at higher levels.

2.2 Outline of Rhetorical Structure Theory

RST is a theory based on work by Mann and Thompson [5] who built on observations by Grosz and Sidner [6]. It basically assumes that documents have a structure. In particular this structure is designed for the purpose of passing on information from the writer to an audience. This implies that sections of the document (sentences, paragraphs etc.) have relations between themselves and a part in the overall purpose of the document. For example one sentence may contain a statement (either fact or opinion) and following sentences may give justifications, attribute the statement to a source, give examples or counter-examples for that statement. Further sentences may then elaborate in more detail, or describe the consequences of that statement.

This structure can be seen in the proceeding paragraph, where the statement is *"This implies that sections of the document (sentences, paragraphs etc.) have relations between themselves and a part in the overall purpose of the document."* Attribution for this statement is given by the proceeding sentences, an example of the statement follows and is elaborated on. Finally in the subsequent paragraph an argument for accepting the statement based, recursively, on the statement is given[6].

It is also assumed that some segments are more central than others. In a relation a central part is labelled **Nucleus**, and a more peripheral part is labelled the **Satellite**. In a list every item may be considered equally important in which case they will all be nuclei but an example would be considered a satellite of the item of which it is an example. In Grosz and Sidner's work they developed a list of 23 different relations some of which would have a nucleus and satellite others of which would have only nuclei.

The particular implementation of RST used in this deliverable is based on the work by Marcu [2]. This follows from the work above and aimed to use computers to automatically locate document sections and label the relations between them. Marcu devised an extended set of **relations** (around 103) and **cue-phrases** (461), words or short phrases, that he believed indicated the relation between a particular section and a preceding or following section. Using punctuation and other surface features, he split documents into their component parts and using the cue-phrases he found in these components, rebuilt the document into a tree structure. An example of which is given in Figure 6.

[6] This explanation is based partly on the mathematical process of proof via induction and partly on the following quote: "If in doubt, make it sound convincing." -Albert Einstein

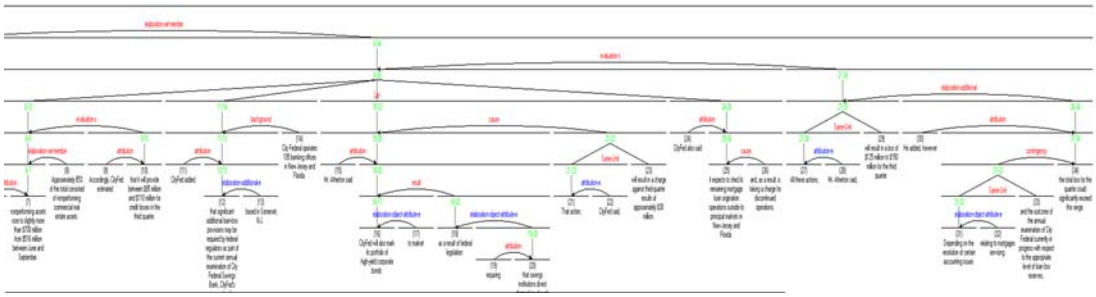

Figure 6 Example of a RST labelled document.

The major problem with Marcu's work is that large parts of it relied on hand crafted rules. In this deliverable it was intended to design a system that could automatically annotate a document without human intervention and without using handcrafted rules, preferably in a language independent way. It was further hoped that this system would be capable of learning. This would mean that should a user wish to annotate documents in other languages, or specific domains that did not respond well to generic English RST analysis, they could provide one or two resources, run a machine learning tool on them and then start automatically annotating without further manual intervention.

This process is described in sections 2.3-2.6.

2.3 Clause Finding

In the original Marcu paper [2] the elementary unit for RST was the clause, rather than the sentence. Roughly this was defined as being the smallest part of a sentence that made sense on its own. Hence clauses could be just the contents of brackets and boundary splits either side of a comma were quite common. Detecting clause boundaries was done by handcrafting rules based on commas, punctuation and cue phrases. Currently experiments are proceeding using a sentence annotator rather than a clause annotator as this is slightly more language independent. The sentence annotator looks for full stops followed by white-space. A list of common abbreviations can be included to avoid interpreting full stops after abbreviations (such as e.g. or Mr.) as sentence endings. The abbreviation section does not have to be included but does improve performance.

Should time be available it may be desirable to write a Marcu style clause annotator but currently it is not felt that the benefits this would bring are worth the time it would take to construct. Additionally, in order to duplicate Marcu's results this would involve generating a list of cue phrases and implementing a very specific human designed set of rules. This rather defeats the object of building an automated RST system.

2.4 Relation assignations (Cue Phrases)

At its simplest associating relations with particular document sections will involve looking for cue phrases. When a cue phrase is found it will be compared to data collected from a corpus to see which relations the phrase can indicate. Additional information can be collected such as whether the relation would be in the preceding or following section or how close that relation might be (e.g. relations between

neighbouring sentences or neighbouring paragraphs). Additional information may include the section's location in the document and the possible relations of neighbouring sections. Putting all this information together should allow a list of potential relations to be produced with a corresponding set of likelihoods for each relation. The most likely can then be chosen.

2.5 Tree Building

At present data is collected about what level in the RST tree structure particular cues are found. This can be used to bias the construction of new trees when annotating documents. At present however it is simply used to rule out infeasible relations (i.e. ones which never appear at the highest/lowest levels). It was intended to investigate better methods in year 3 after ascertaining the performance of relation annotating.

2.6 Rule Learning Algorithm

This is the general process being used for training and classification. However it was intended to try modifying the various "likelihood" scores used to try to more closely simulate human judgement in annotating documents. Hence at this stage in the project, the overall process being used is important rather than the specific calculations of likelihood at each step.

2.6.1 Training

A set of training documents is processed. Each training document is split into fragments (clauses or sentences) and a relation tree is defined by human annotators that joins the fragments. This tree is usually binary branching. The tree is considered to be a set of nodes. The leaf nodes simply consist of a fragment of text. All nodes above the leaf nodes join two lower level nodes in a relation.

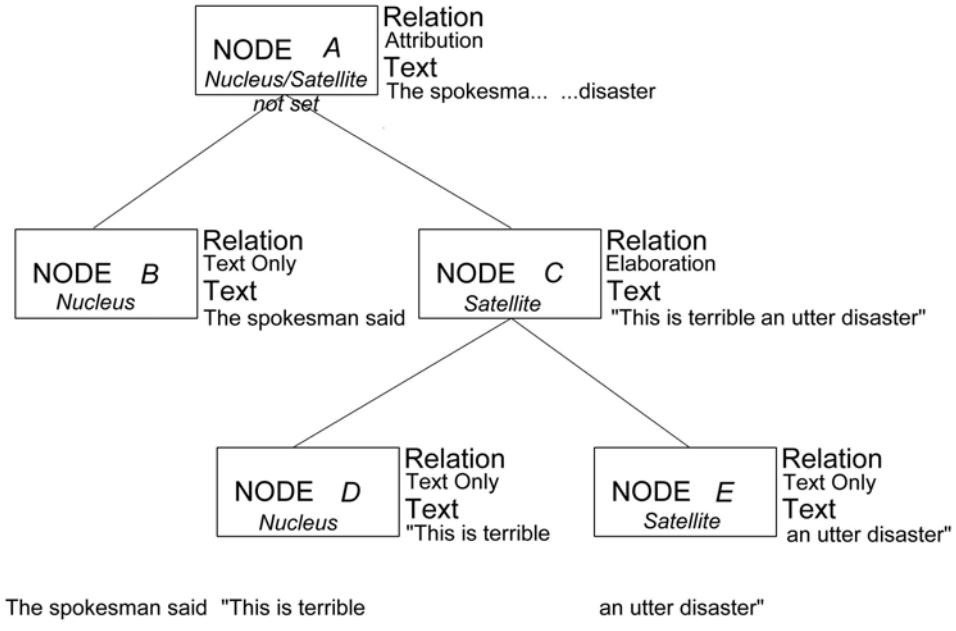

Figure 7 Node layout diagram

Each relation node is processed, and all cue phrases found in the text spanned by its left branch are collected. For each cue phrase in turn the combination of the cue and the relation is used as an index. Counts of their frequencies are stored, along with the presence or absence of leaf nodes and the nuclear / satellite status of the branch, immediately below the relation node. The nuclear/satellite status of the right hand branch along with the presence or absence of leaf nodes immediately below the relation node is also stored separately, as the cue could be considered as giving evidence for both halves of the relation, not just the half it appears in.

For example in Figure 7 Node layout diagram the cue "said" is counted as appearing in node B, a nucleus, in the left hand branch of an *attribution* relation at the lowest possible level. The right hand branch of this relation (node C) is counted as being a satellite of an *attribution* relation appearing immediately after a node containing the cue "said" at an intermediate level. If Node A is the root node of the document then this is the end of the process, otherwise the cue "said" is also noted as appearing in node A and whatever relations node A forms at higher levels. Thus it can be seen that a single cue can be counted on multiple levels.

The same processing is then carried out for all cue phrases found in the right hand branch.

Nodes that do not contain any cues are treated as though there was a cue named "NULL" in them and counts collected in the same fashion as for all other cue phrases.

Once these counts have been found specific conditional likelihoods[7] are calculated in general they are:

$$\ln\left(\frac{count(relation \mid cue)}{count(cue)}\right)$$

The use of ln() transforms the likelihood range of between 0 and 1 to a range between 0 and –infinity, the closer the number is to 0 the greater the probability. This range increase helps prevent some rounding/overflow issues when adding and multiplying very small numbers. The rest of the formula comes from Bayes Theorem

$$prob(a \mid b) = \frac{prob(b \mid a)}{prob(a)}$$

2.6.2 Annotating

Step 1

The document is split into text fragments and the cue phrases found in each fragment are collected. Scores are collected for each fragment, based on the conditional probability of it being the left branch/right branch of relation X given the cues that appeared in the fragment itself or the fragments immediately to its left or right. Its likelihood of being a nucleus or satellite is similarly calculated along with its likelihood of appearing at a high or low level in the tree. A series of these combinations is stored for each fragment along with the associated likelihood for that combination. Fragments without a cue are given likelihoods based on the "NULL"

[7] These are likelihoods as they are not between 0 and 1 and therefore cannot be probabilities. They do more or less the same job though.

cue counts found. Each text fragment then has a node dedicated to it. The node spans only that fragment and is a "leaf" node not containing any relation. The likelihoods attached to the fragment are now attached to the node.

Step 2
After the likelihoods have been collected the highest likelihood combination for each node is found (e.g. it is the nucleus and start node for a "attribution" relation with the fragment that follows it at a low level in the tree.) . The combination is then checked for feasibility (for example it cannot be the first node in the document and the second node in a relation). If infeasible then this combination is deleted from the list for that node and step 2 is repeated.

Step 3
The combination with the highest likelihood is then used to combine two nodes together, the two existing nodes, A and B, are labelled with the appropriate Nucleus/Satellite designations. A new node, C, is created and Nodes A and B register C as their parent node. Node C is then given a relation, and stores A and B as its child nodes, it also stores the span of the text it covers which is the sum of the text covered by A and B.
The likelihoods for nodes A and B are then passed up to node C. Where the same combination existed for A and B separately the likelihoods are summed together[8].

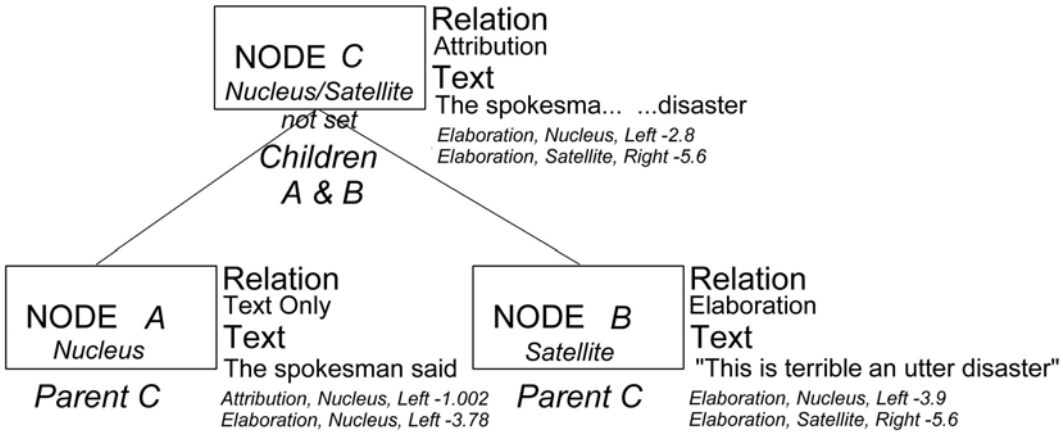

Figure 8 Node creation example

A and B are removed from the list of nodes to be processed and C is added to this list. If node C now spans the entire text the process stops, otherwise it returns to step 2.

[8] As we are using likelihoods the highest likelihood is the negative number closest to but not equal to 0. Additionally when summing two likelihoods the process involves raising e to the power of each likelihood, adding the results then taking the natural logarithm of the sum.

2.7 Current Module Performance

2.7.1 Evaluation Method

A series of strictly binary RST trees that have been hand created by trained human annotators has been held out from training. Text files containing the un-annotated text from these trees have been obtained with line breaks inserted where the human annotators felt a text break was appropriate. When processed this will group text into the same fragments used by the human annotators.

As each fragment N has some relation with fragment N+1 (due to the binary nature of the trees produced), we can see if this relation matches the human provided relation. We can separately score whether the Nucleus Satellite labels are correct. It scores 0.5 for each correct branch, so getting both right gives a score of 1. Due to the existence of Nucleus/Nucleus relations each branch has an independent chance of being right or wrong and it was considered worthwhile to collect these chances separately.

If the comparison is restricted to inputs that have only binary branching trees then we can compare what percentage of branches are over the same two spans. These will be considered correct if both branches match, and incorrect otherwise.

This will give 3 scores, 1 for relation matching ability, 1 for finding nucleus/satellite positions and 1 for the tree structure. Although the idea of having a single metric is appealing this division of output wil show more clearly where the system is working well and badly (errors in tree construction will not hide successes in relation finding or vice versa) and though it is expected that improvements in one area will increase scores in the others it will help show where the majority of effort is needed.

Ideally two variants of each of the scores would aid comprehension.
One score would be average percentage per document:

$$\frac{\sum_{N} \frac{Correct_{doc}}{Total_{doc}}}{N} \times 100$$

where N is the number of documents, $Correct_{doc}$ and $Total_{doc}$ are respectively the number of correct annotations and the total number of annotations in a specific document.
The second varient would sum all correct annotations across all documents and divide it by the total possible number of correct answers:

$$\frac{\sum_{N} Correct_{doc}}{\sum_{N} Total_{xoc}} \times 100$$

The second is the more important measure to improve long term but the first will prevent high scores on small trees or one atypically poor result on a large tree giving a misleading overall picture.

2.7.2 Results

These are the results across 38 documents having an average of 61 relations per document.

	Percentage Expected by chance	Percentatge across all relations	Average "per document" percentage
Relation Name	1	9.8	11.06
Matching branch spans	$5.6x10^{-4}$	3.35	6.66
Nucleus/Satellite designation	50	45.4	46.6

The sheer number of relation classes used by Marcu leads to major data sparcity issues that appear to be crippling the system. The difference in "per document" and "all relation" scores indicate the system does better with shorter documents. It can be seen that selecting the relation names and the branch spans are working at a level far above chance (so the method used "works") but not far enough to be useful. The failure of Nucleus/Satellite detection is something of a puzzle. Overall it is a fairly gloomy picture.

2.8 Research Directions

Now the evaluation software is in place the next step would be to proceed in examining data sparsety. The current test set (circa 300 pre-annotated documents) could be divided into sets of 100-300 examples in 50 example increments; the results from training and testing a system using these reduced training sets plotted on a graph. If the results are on a straight upward slope this would indicate that data is the major limiting factor. If the results are uncorrelated then our current method is a failure and work should terminate. If the results are in a log style curve (i.e. futher improvement through more data would require unfeasible amounts) then the best way to get improvement is algorithmic modification and would be a justification for more research. Currently it is felt that the results would indicate that there simply is not enough data for the training software to work at a level sufficient to be useful with the current set of classes. If the results of the data sparsety test indicated otherwise potentially more data could be created.

It is suspected the size of the corpus would have to at least double to provide sufficient raw data. Assuming a mark up time of 1 hour per document (based on prior attempts) then doubling the size of the training corpus would take 300 man hours. Since this would probably have to be done with newspaper articles from a different source an additional 30-40 man hours would be needed to expand the testing corpus by a proportionate amount.

An alternative would be to move onto grouping the relations into a small set of common, or easy to find relations that can be consistently marked up. For example all relations that occur more than a thresholded number of times in the training corpus with all other relations being grouped into an "other" relation. Or grouping similar

relations together in "superclasses" it may be possible to detect this smaller number of relations with greater accuracy without increasing the amount of data in the training corpus. However this would require algorithmic improvements for organising the tree structure of the system and the selection of nucleus and satellite relations, as no extra data would be provided to aid these functions.

2.9 Conclusions

The first years work was creating the infrastructure to automatically annotate documents with discourse structure and deliver the annotated document in some useful way. This has been achieve and a system produced that provides 3 different access methods and 4 different output formats that should allow BT internal and external users all the flexibility they need and has proved more than satisfactory when implementing the first classifiers.

This years work has proved somewhat disappointing. While the framework established worked as expected (indeed has remained totally unchanged from an external perspective, though has been recoded internally for additional speed, maintainability and reliability) the problems of automatic annotation have proved troublesome giving results far above those expected by chance but far below those required for any realistic task.

Simply put the data used (Marcu's corpus of hand annotated documents [1]) has proved insufficient to allow the automatic categorisation of the more than 100 different relation types. Potentially work could continue by combining the large number of relations into a smaller group of more broadly defined classes. However this conflicts somewhat with the hands off approach originally envisaged. Alternatively more data could be created but this would be likely to take a prohibitive amount of time. Estimates suggest circa 2-3 months of full time effort. As this project has only half a person on it this would obviously double by "calendar" time.

A judgement needs to be made as to whether the reduced and diluted aims now envisaged justify the remaining years work. SEKT task 2.3 should probably be chalked up as worthwhile research that simply returned a negative result. To paraphrase Thomas Edison – "This isn't a failure. I've simply found another 100 ways that don't work."

3 Bibliography and references

[1]

[RST Corpus 2002]
RST Discourse Treebank 2002
ISBN: 21-58563-223-6
Lynn Carlson, Daniel Marcu, Mary Ellen Okurowski
http://wave.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2002T07

[2]

[Marcu 1997]
The Rhetorical Parsing, Summerization and Generation of Natural Language Texts
Daniel Marcu
Thesis, University of Toronto (1997)
http://www.isi.edu/~marcu/papers.html

[3]

[Gate System]
General Gate pages
http://gate.ac.uk/
Download from
http://gate.ac.uk/download/index.html

[4]

[Axis 1.1]
General Axis pages
http://ws.apache.org/axis/
Download Java libraries from
http://ws.apache.org/axis/java/install.html

[5]

[Mann 87]
Rhetorical structure theory: A theory of text organization
W. C. Mann and S. A. Thompson
The Structure of Discourse, ed., L. Polanyi, Ablex Pub. Corp., Norwood, N.J., (1987)

[6]

[Grosz 86]
Attention, intentions, and the structure of discourse
Barbara Grosz and Candice Sidner
Computational Linguistics, 12, 1986

4 Index

Clause
Smallest section of a text document individually marked up by RST algorithms. Intuitively the breaking up of a sentence into its components, algorithmically found by breaking up the sentence based on commas, brackets, speech marks or specific cue phrases.

Cue phrase
A word or phrase that has been identified as being connected with either some sort of sentence break or consistently connected with a particular relation. For example “for example” usually leads to the following text being an example of a concept introduced by the previous text.

Deep parsing
A generic term used in language processing. It is usually used to describe a process of analysing a document computationally at the level of word meanings. Deep parsing the sentence “Sam kissed Tom.” would indicate that there was an entity “Sam” who “kissed” an entity “Tom”. Further deep parsing could involve inferencing that as “kiss” occurs between humans, Sam and Tom are human. That there was an event “kiss” that is being described and it happened in the past etc.

Nucleus
When document sections are related the “more important” section(s) is labelled the nucleus. If X is an elaboration of Y, then Y would be the nucleus.

Relation
Effectively a human meaningful label that is used to indicate a specific type of connection between two or more sections of a document.

Satellite
When document sections are related the “less important” section(s) is labelled the satellite. If X is an elaboration of Y then X would be the satellite.

Surface feature
Features directly derived from the string of characters that make up a document. Defining a word as “a group one or more of the characters [a-zA-Z] with a leading and trailing space character” would be finding words via surface features.