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

D2.3.1 Deep Structure Analysis V1

James Allen (BT)

Abstract
This document accompanies the D2.3.1 software deliverable and contains two major sections. The first describes the architecture, design and usage of the D2.3.1 deliverable. This document describes software which 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 the research directions that will be taken to develop the more advanced versions of this module that will be supplied in D2.3.1 V2 and V3.

Those only interested in using the module as a black box can of course ignore section 2 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
December 2004

PU/PP/RE/CO
December 2004

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
Contactperson: 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
Contactperson: 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
Contactperson: 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
Contactperson: 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
Contactperson: 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
Contactperson: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Intelligent Software Components S.A.
Pedro de Valvidia, 10
28006
Madrid
Spain
Tel: +34 913 349 797
Fax: +49 34 913 349 799
Contactperson: 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
Contactperson: Tom Bösser
E-mail: tb@keapro.net

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

Sirma AI EOOD (Ltd.)
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768, Fax: +359 2 9768 311
Contactperson: 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
Contactperson: 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
Contactperson: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

This document is in two parts. The first is a guide to the use of a 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 future research directions over the next 2 years of the SEKT project.

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
- Paragraphs 2 & 3 elaborate on the subject introduced in paragraph 1
- Paragraph 3 is a shift in topic from paragraph 2

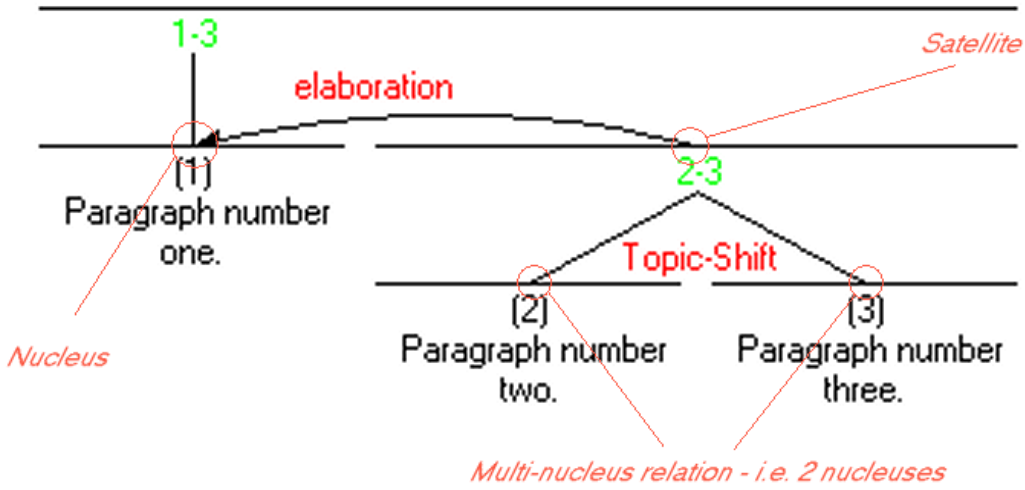

Figure 1 The RST tree for the first 3 paragraphs of this document.

The information gained from a RST parse is usually provided in a tree structure (see Figure 1) with nodes being labelled with the relation connecting the branches (usually 2 of them) coming from it. Individual branches are either a "nucleus" or a "satellite" depending on whether they are considered to be of primary or secondary importance in their particular relation.[1] 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 – such as punctuation or certain cue phrases "as follows", "alternately" etc. - are used, RST can potentially analyse documents in real-time 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.

[1] In diagrammatic form most relations are shown with an arrow pointing from the satellite to the nucleus. If there is no arrowhead the realtion is a multi-nucleus relation and all branches have the status nucleus.

In the first year of the SEKT project the general RST parsing approach described above was used to create a fully automatic, but limited, system. 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.

This automated system and this document [D2.3.1] were the output of the first years research. Most of this research was concentrated on the practical details of understanding and implementing some form of RST system. In particular this involved solving problems related to providing the system to users of GATE and also the those who wanted to use the system directly. As such the current system is more of an implemented architectural design to iron out communication and processing issues and to test potential output formats for the RST trees than a full attempt at accurate RST parsing. 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.

When this architecture had been developed a very simple classifier was implemented to test its basic functionality. Little effort was made to get good results from this classifier. As such the RST trees produced are very poor. At the level of the relation between neighbouring clauses the system was basing its decisions, to some extent, on data from a human annotated corpus but the higher level relations and the choice of which branches to merge were taken arbitrarily and as such the results were not evaluated as they were expected to be that of chance. The second and third years are to be spent researching and implementing efficient and high performance RST algorithms now that it is known that the framework is practical and effective.

The research to be conducted over the next two years will focus on developing a fully automated system in contrast to current "automatic" annotators which usually seem to require large amounts of human tuning and intervention. It is intended that the system should be given a training corpus (in any language) and possibly a list of potential cue-phrases. With these two resources it should be able to train itself and to carry out full RST parsing.

Contents

1 Architecture, Design and Use of D2.3.1

1.1 Introduction

This document forms part of the deliverable D2.3.1 for the SEKT project. In it 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) 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**[2] 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 in 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.

[2] A clause in this case is the level below a sentence and could include the contents of brackets, the parts of a sentence before a comma or the contents of quotes. Its exact definition varies and the SEKT software currently uses sentences instead.

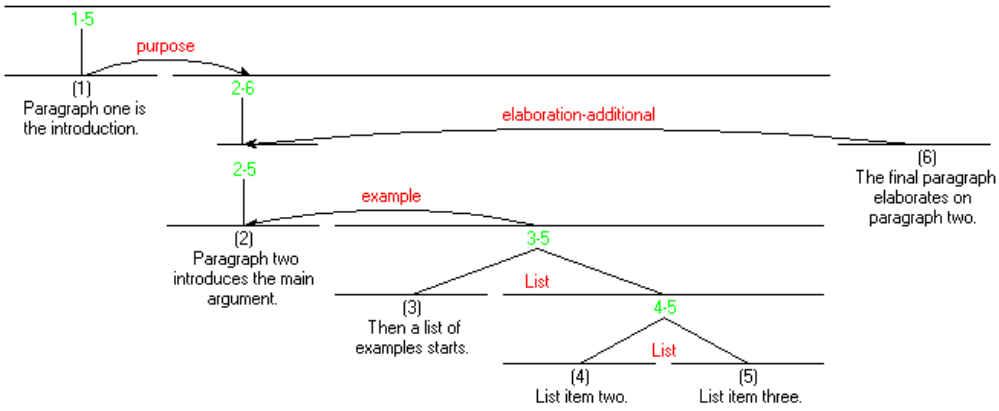

Figure 2 Example RST tree.

The example tree in Figure 2 shows this diagrammatically. Most relations have a central nucleus and a satellite with an arrow that points to the nucleus.

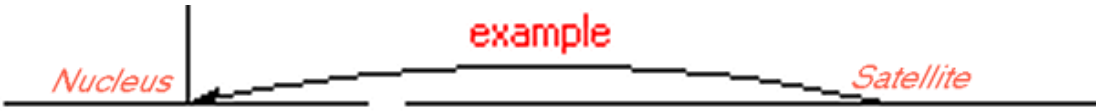

Figure 3 Nucleus / Satellite Relation

Multi-nucleus relations are shown as pairs of equally important relations.

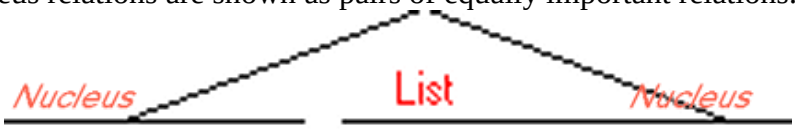

Figure 4 Multi-Nucleus Relation

The tree shown is slightly different from a standard RST tree in that the text components are paragraphs rather than clauses but the basic structure is identical.

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

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 D2.3.1 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 will take a document in text or html format as input and return a set of annotations. These annotations will label either leaves or nodes. The leaves will span a piece of text and will contain the label for that text (a number between 1 and N) and the id of the text's parent. The nodes will 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 nucleuses or satellites. From this information the RST tree structure can be built up in whatever format best suits the objectives of the module user.

1.2 System Design and Architecture

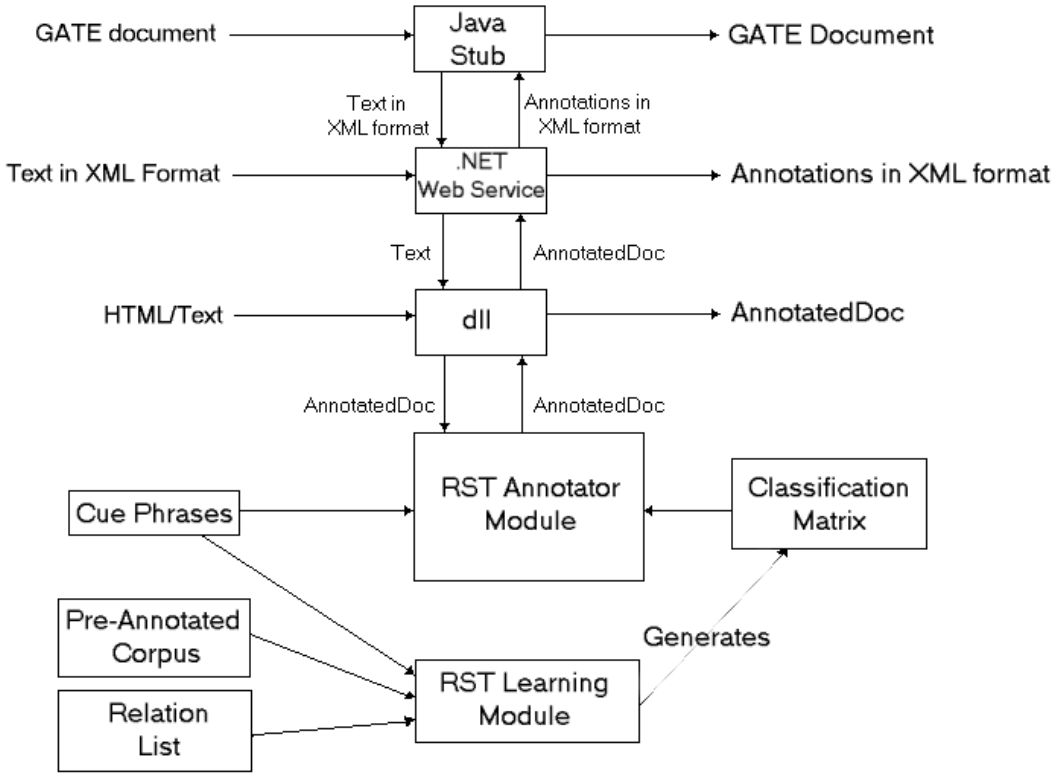

Figure 5 Module design architecture

The architecture shown above in Figure 5 is built around some of BT's existing language processing software which is somewhat analogous to Sheffield's GATE system, sharing several features in common. 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 pointers have a numeric value which is the offset of that point, i.e. number of characters in the document that occur before the pointer. 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 with 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 indicated. This information is used to select the most likely relation. For a more detailed explanation of this annotation process see 2.4 Relation assignations (Cue Phrases) onwards. These annotations 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. 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 System.IO;
public class Tester {
  public Tester() {}
  public static void Main(string[] ARGS) {
    ParseDoc pd=new ParseDoc();
    pd.setMatrix(@"c:\GATE\RSTCorpus\fullMatrix.txt");
    pd.setCueSet(@"c:\GATE\RSTCorpus\cuePhrases.txt");
    // alternately readHTML or readXML handlers could be used.
    ReadTXT rt=new LTG.AnnotatedDocReaders.ReadTXT();
    FileStream fr=new FileStream( @"C:\GATE\RSTCorpus\wsj_1384.txt",
                                  FileMode.Open);
    AnnotatedDoc ad=rt.ImportDoc(fr);
```

```
    pd.getRST(ref ad,"RST",@"@C:\GATE\RSTCorpus\CuePhrases.txt");   }
    // use ad
}
```

The document would then be contained in the AnnotatedDoc ad with a set of annotations that could be accessed. For details of how to use this see the AnnotatedDoc documentation.

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 6 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[3].

```
<?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 7 Example output from Web Service

[3] 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[4] 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"));
```

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

Of course this can be done most easily by doing a search and replace on “oldhost”. 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[5] [4] both of which can be downloaded from the internet and have extensive documentation on the download sites.

[5] 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 and deductive logic to generate methods of extracting meaning from a given text founded on linguistic theory. In general the approach is labour intensive and the results often disappointing with systems proving brittle or domain specific.

The second is to make a set of simplifications to the problem that allow the use of statistical tools. The problems then become choosing a feature set of some description and collecting sufficient data to learn the correlation between collected features of the document and the desired output. This usually involves hand annotating a corpus with the desired output and then running some form of learning algorithm over the annotated corpus. Both of which can be done in limited time with limited linguistic knowledge This is usually quite quick and often works better than the more linguistically motivated techniques. The underlying reasons for this are unclear[6].

This particular deliverable did not have access to any large knowledge bases or the significant number of person-hours that would be needed to manually generate rules for the discourse analysis of text. All of which suggested that the second approach would be preferable. Additionally linguistically motivated approaches are notorious for being limited in domain and usually language dependent. As the project aim was to develop annotation tools that are robust across arbitrary domains, and a secondary aim to make the resultant software language portable, this mandated a more statistical approach. Finally previous work at BT on other language processing tasks had built up a knowledge base of statistical techniques that could be applied to this project.

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, e.g. mapping the word bank to either "high street money managing institution" or "interface between a river and the land on either side" dependant on the syntax (the letters bank/BANK/Bank etc.) and the pragmatics (whether the surrounding text refers to money or countryside). When the concepts have been derived they can be used 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 unsuited to real-time processing) and easier to implement. Additionally shallow techniques are usually more robust (i.e. less liable to

[6] "If we knew what it was we were doing, it would not be called research, would it?" --Albert Einstein

fail when applied to new domains) which is important for software used as part of a process of document annotation where having a system that always returns a result, of some level of accuracy, is better than a system that has a higher average level of accuracy but sometimes fails entirely.

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 and does not need in-depth linguistic knowledge for its use or 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 preceding 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[7].

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 nucleuses but an example would be considered a satellite of the item it is an example of. 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 nucleuses.

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 and **cue-phrases**, words or short phrases, that he believed indicated the relation between a particular section and a preceding or

[7] 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

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 8 below.

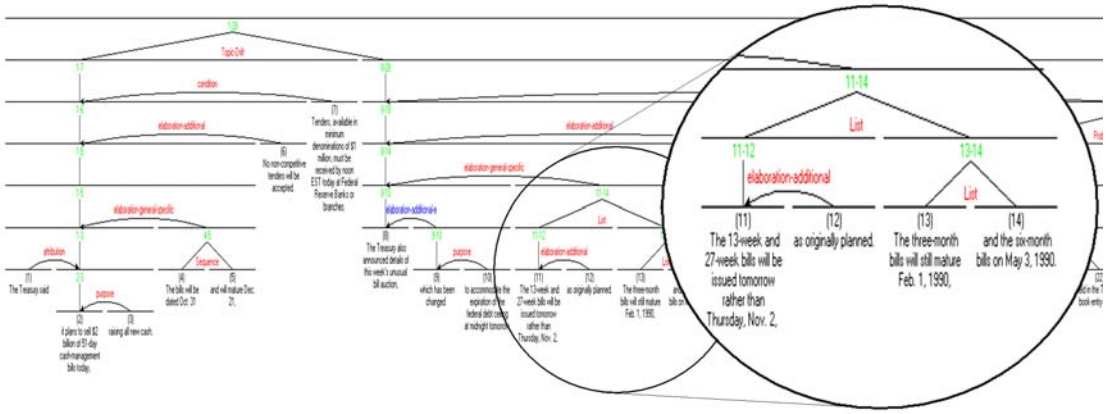

Figure 8 Example of a RST labelled document.

Marcu's work has the advantages of being a relatively simple and intuitively appealing theory of document structure which lends itself to automatic processing. The major problem with Marcu's work is that large parts of his implementation 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. 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 putting full stops after abbreviations such as e.g. or Mr. 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, which 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

Tree building is the area where some research is still needed. How to tell at what level document sections should be joined together is an open question. The first direction to be explored will be to collect all possible relations at the lowest level, select the best and then "bubble up" the relations that were not selected to be used at higher levels. In other words bottom up tree creation. Further research will depend upon the results of the experiments here.

2.6 Rule Learning Algorithm

Currently the algorithm is a fairly simple one that reads in a corpus of pre-annotated documents, counts co-occurrences of cue phrases and relations and then weights the co-occurrences with the number of times the cue phrase occurs in total. This gives the co-occurrence rate.

When annotating a new document the algorithm simply looks for cue phrases in each sentence in turn. If it finds the relation with the highest co-occurrence rate with that cue phrase and returns that.

At this point the actual learning algorithm stops and an arbitrary process starts. The final pair of sentences are considered to be related, one is chosen (randomly) as a nucleus and its relation chosen as the relation joining the two sentences, the relation from the satellite that was not selected is passed up. The previous sentence is then joined to the new relation, again a nucleus is chosen randomly and the created relation given the nucleus's label. This process repeats until all sentences are combined together.

2.7 Current Module Performance

It is not currently useful to evaluate module performance. As the tree building sections have not yet been implemented the only relations that could be usefully compared would be at the lowest (i.e. leaf) level. It was decided that writing testing software specifically to do this was not a productive use of time. When some form of tree construction is implemented software will be written to compare manually and automatically produced trees for particular documents. This software will produce precision and recall figures. This will be high priority for next years deliverable.

2.8 Research Directions

The research plan going forward effectively consists of three steps:

The main area of research is going to be how to organise the overall structure of relations above leaf level. Determining relations between neighbouring sentences seems relatively straightforward. However joining the related sentences into larger

clusters, and these clusters into even higher level relations, leaves the problem of determining which relations apply at which level. This is the primary focus of the next deliverable.

Once some form of tree structure can be built it then needs to be evaluated and development of evaluation software will be the next step in the research program once the annotation software can produce full trees based on learned rules.

Finally when these two issues have been overcome then the research will focus on collecting and using different features of the training data and investigating changes in the performance of the automatic annotation using different feature sets.

2.9 Conclusions

The main focus of this 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 achieved and a system produced that provides 3 different access methods and 4 different output formats that should allow users all the flexibility they need. It has been designed with modularity and experimentation in mind and can be quickly updated with newer classification techniques.
The automatic generation of discourse information is a difficult task. Currently the module provided will only give partial information. However the format it returns this information in should remain constant for the lifetime of the SEKT project. Therefore the module as it stands should be useful for integration purposes and when deployed as a web service future advances in annotation accuracy will be automatically incorporated by users.

The methods applied to this task are fully automatic once trained. In addition anyone able to provide defined data (a list of potential cue phrases, a RST corpus) could train a new RST annotator in a different language or an annotator specifically optimised for a particular domain. While this data may not be widely available it at least has the merit of being well defined and avoids the need to have trained computational linguists who can hand-generate rules for modified circumstances.
In summary the first year of work on this project has produced a firm foundation for future experimentation and should deliver an excellent base for the next two years of work in this area.

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.