AUTOMATICALLY GENERATING TEXT TO ACCOMPANY INFORMATION GRAPHICS

by

Mary Ellen Foster

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Automatically generating text to accompany information graphics

Mary Ellen Foster

Master of Science

Graduate Department of Computer Science

University of Toronto

1999

Generally, when quantitative information is to be presented, some form of graphical presentation is used, often with a textual caption to ensure that the audience notices particular aspects of the data.

This thesis presents the principles that should be followed by a system aiming to produce such captions automatically. The process of caption generation is examined in the context of the standard tasks in text generation. Most previous systems in this area produce textual summaries intended to stand alone; the issues involved in producing a caption differ, as the text must be coordinated with the graphic it is to accompany. The thesis also presents CAPUT, a prototype caption-generation system which follows these principles to generate single-sentence captions for information graphics of the type that might appear in a newspaper article. Finally, extensions to CAPUT that would bring it from a prototype to a full-fledged caption generation system are proposed.

Acknowledgements

It seems to be necessary to mention that a picture is worth 1000 words somewhere in any thesis in the area of multimedia generation—consider this my mention. With that out of the way, here are a few words of my own to thank people who have been helpful with this thesis.

Thanks to my supervisor Graeme Hirst for keeping me on track and for his excellent advice and proofreading skills. Thanks also to my second reader, Mark Chignell, for taking the time to read and comment on this thesis.

Thanks to all of the visitors to the Computational Linguistics group during the past months for the opportunity to discuss my research; many new ideas came up during these discussions.

Thanks to Marc Corio and Sarah Boyd for giving me early access to their respective theses.

Thanks to my coaches and classmates at the University of Toronto Gymnastics Club; to the Starfleet Ladies' Auxiliary and Embroidery/Baking Society; and to all my other friends in Toronto, Waterloo, and elsewhere for helping to keep me sane.

Finally, a couple of special thank-you notices...

Thanks to Ron, my personal cheering section (Yay!) and my best friend.

And thanks to my parents: to my father Leslie for introducing me to the joy of computers, and to my mother Margaret for introducing me to the joy of language. Just look what you've started...

Contents

List of Figures

Chapter 1

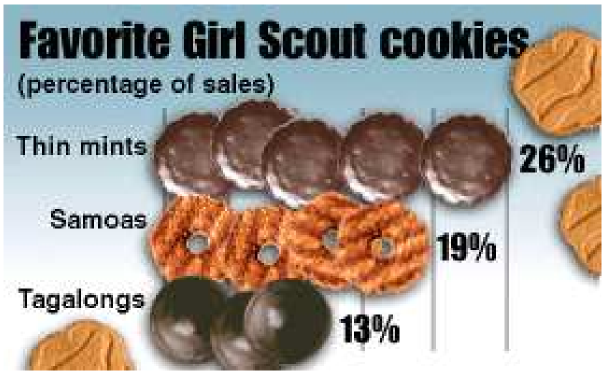
Introduction

1.1 Presentation of quantitative data

Presenting large amounts of quantitative data effectively is not an easy task. Depending on the situation, the same data can be presented in a variety of ways—different messages may be extracted from the data, or those messages may be presented using different techniques. The author of the presentation may want simply to inform the audience, or he or she might have the goal of persuading them of a particular fact. A skilful author will make sure that any data included in a presentation is displayed in the most effective way, using appropriate techniques so that the desired point is illustrated.

Choosing appropriate presentation techniques is a difficult and time-consuming task. For example, every five years, the government of Canada performs a national census. The raw data from this census is made available within months of its completion; Statistics Canada then spends the next several *years* producing hundreds of publications which present a variety of analyses of that raw data (Corio, 1999).

Figures 1.1–1.3 show various different presentations of quantitative information. Figure 1.1 is taken from the newspaper *USA Today* (USA Today, 1999). It is an example of the "USA Snapshot" ("a look at the statistics that shape our lives") which is printed on its front page each



Monster cookies: Thin Mints cookies are tops to Girl Scout cookie consumers.

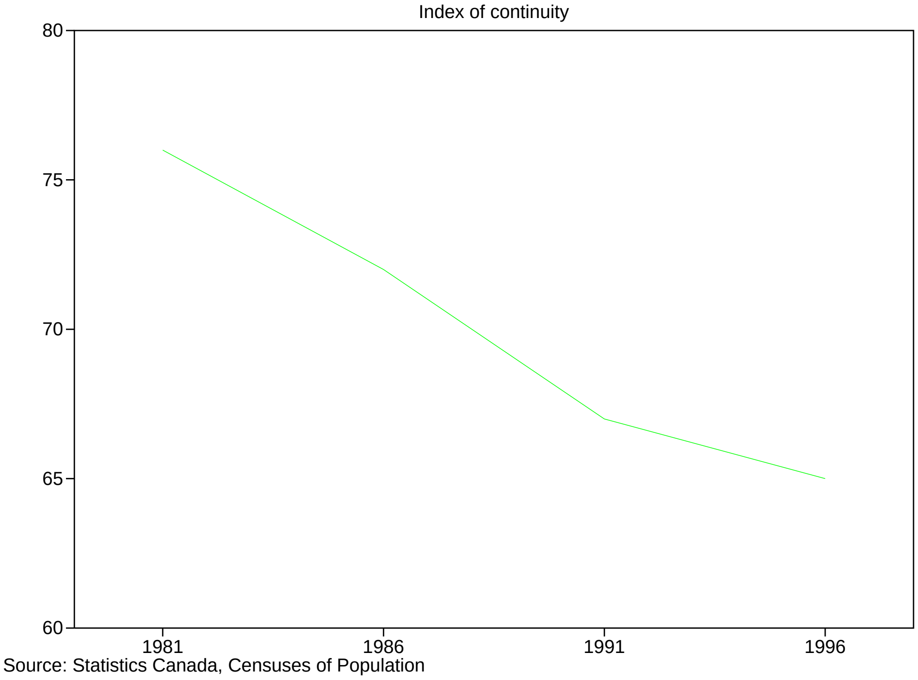
Figure 1.1: Bar chart and caption from *USA Today* (USA Today, 1999)

day. The goal of this graphic is simply to inform the audience of some interesting fact; there is no effort to persuade or convince. Notice that the caption concentrates on the value of only one of the variables, while the graphic presents the rest of the data itself. The style of the graphic, with its gimmick of using images of cookies to make up the bars, fits well with the casual tone of its caption.

Figure 1.2 is adapted from an article in a Statistics Canada publication (Norris, 1998). The article, written by a statistician, discusses the findings about aboriginal language use from the most recent national census.[1] As in the previous example, the presentation was created with the simple goal of presenting a fact to the audience; once again, no attempt at persuasion is made. In this case, though, the overall trend of the data is described, rather than a single data point as in the preceding example. As well, the tone of both the graphic and the caption are quite different, reflecting the difference between the respective contexts of the two presentations.

The third example, Figure 1.3, shows a graph and caption adapted from a report to mutual fund investors (Fidelity Investments, 1998). The goal of this article is to convince investors that staying in the stock market is worth it in the long term (and that they should therefore choose

[1] The "index of continuity" displayed in the graphic measures the vitality of a language by comparing the number of those who speak a given language at home to those that learned it as their mother tongue. The lower the score, the greater the decline.



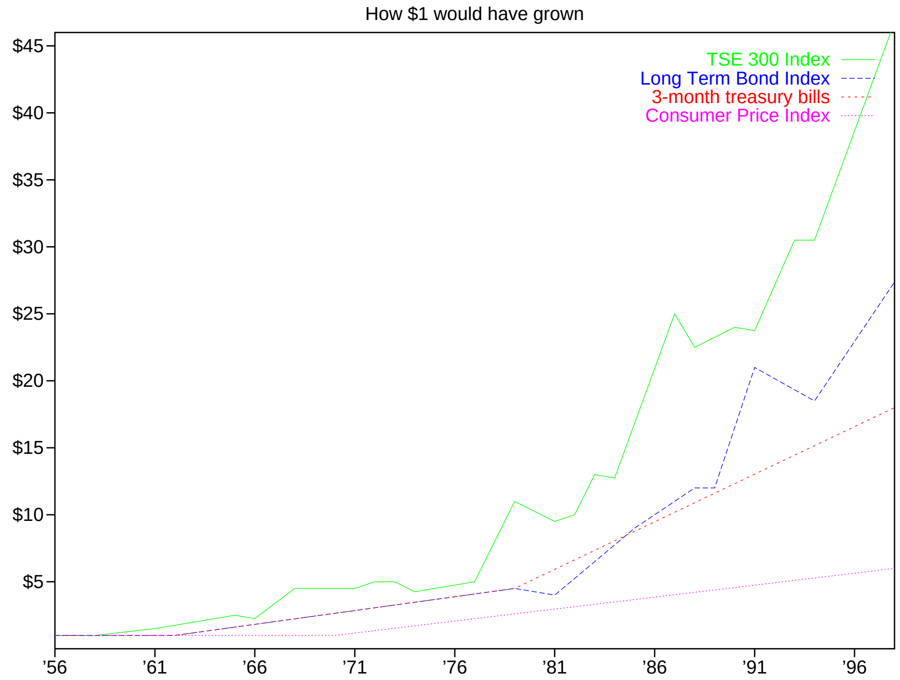
The index of continuity for Aboriginal languages has declined steadily over the past 15 years.

Figure 1.2: Line graph and caption from Statistics Canada publication (Norris, 1998)

funds which invest in stocks). Notice that in this case, as in the first, the value of one of the variables is singled out; however, here it is an entire trend which is singled out. In the more complex data presented in this example, many other aspects of the data could have been presented; however, in keeping with the goal of the article, the trends of the TSE 300 line were emphasized. Section 1.2 examines the issues of the textual components of presentations in more depth.

1.2 The importance of text

The presentations in Figures 1.1–1.3 contain graphs of the sort typically used when quantitative information is to be presented. Figures 1.1 and 1.2 contain fairly simple data, so the text in the caption is also straightforward. However, Figure 1.3 is more complex. If we ignore the graph's caption and concentrate only on the lines on the graph, many different features of the data are apparent. For example, the TSE 300 Index obviously had the largest increase over the time



Each type of asset performs differently over time. Consider that $1 invested in stocks represented by the TSE 300 Composite Stock Index, assuming all dividends were reinvested, would have grown to $46.80, outpacing long-term bonds and 3-month T-bills by a substantial margin. Note, however, the greater short-term ups and downs.

Figure 1.3: Line graph and caption from mutual fund report (Fidelity Investments, 1998)

period; similarly, the Consumer Price Index increased the least. Also, the growth of Long Term Bond Index and of 3-month treasury bills are quite closely correlated until about 1990, at which point the values diverge. As well, the Consumer Price Index has the smoothest line, while the TSE 300 has the most changes in slope. The feature of interest could also be the steep decline in the TSE 300 in 1986–87, or its subsequent steep rise from 1994 on. Any one of these features could be the message of the graph, depending on the context in which it appears and the point that the author is using it to support.

The original graph upon which Figure 1.3 is based appeared in an article in a report to mutual fund investors (Fidelity Investments, 1998). The title of the article was “History Shows That Stock Markets Have Always Rebounded” , with the caption shown in the figure. The title of the article in which the graphic appeared gives its context, and provides a suggestion of the author’s motivation for including that information. Notice how the caption helps focus the attention of the reader on exactly the aspect of the data that the author desires to emphasize: the fact that the TSE 300 increased by far the most over the time period in question. Without text, the graphic in Figure 1.3 still leaves itself open to several different interpretations.

According to Kosslyn (1994), the text which accompanies an information graphic can serve two purposes: to clarify unfamiliar terms and graphical notations, and to point out specific features of the graph. This text does not necessarily have to be included directly above or below the graphic itself; it can also appear in an article which makes reference to the contents of the graphic. For the purposes of this thesis, we will concentrate on the latter function (pointing out features of the graph), and for simplicity the generic term "caption" will be employed for both forms of accompanying text.

Many authors have dealt with the question of choosing appropriate graphic techniques to present statistical information in a variety of situations—among others, Zelazny (1996) and Kosslyn (1994). These books give advice to presenters on how to create an effective presentation of a variety of statistical data. Similar hard-and-fast rules for creating appropriate textual presentations do not exist, and it is not clear that such rules would be useful.

The caption should complement the graphic without simply listing all of its features—unless, of course, that is what the user requires. Normally, though, the caption will concentrate on one or two significant features of the data as a whole; often, some of the input data items will not appear in the caption at all. Choosing which data features to include in the output and combining the selected data are not always straightforward tasks; they may be affected by many factors, including the nature of the data, the domain, the form of the graphical presentation, and the communicative goals of the user.

As well, the style of the caption may vary greatly depending on its intended audience. Figure 1.1 is a classic example of the sort of graphic and caption that appear daily in the newspaper *USA Today*; its novel graphic and folksy caption are very different from the businesslike presentation of Figure 1.3 and its accompanying text.

1.3 Automated generation of presentations

When a large number of presentations must be produced, such as by Statistics Canada, an automated or semi-automated system to create the presentations is very useful. The presentation rules provided by such manuals as that of Zelazny (1996) can be used as the basis of such a system. PostGraphe (Fasciano, 1996) is one such system; it automatically produces an appropriate graphic based on the data, a characterization of the context, and the desires of the presenter.

To produce full presentations, text should also be generated to accompany the graphics. This text should follow the criteria outlined in the previous section. It should use the same generation criteria as the graphic, and should complement the graphic rather than simply restating its content.

Sometimes, the user has available all of the data and knows what the trends are before the caption is generated. In this case, the specific trends to concentrate on in the caption can be specified beforehand. However, the user may not actually know the nature of the data before. In such situations, the user should specify the general sort of thing to look for, and the generation system will then use rules to choose the specific content of the caption.

The goal of this thesis is to present the factors that should influence the automatic generation of text to accompany information graphics, and to build a system which takes into account those factors.

1.4 Outline of the thesis

Chapter 2 presents some previous work in the automated generation of presentations of quantitative information. A number of related projects in this area are presented.

Chapter 3 outlines the principles which should be followed by a caption-generation system during the process of generating text to accompany a graphic. The tasks which such a system should perform are described, along with other factors that the system should take into account during the generation process.

Next, Chapter 4 describes CAPUT, a particular implementation of a caption-generation system. It is implemented in Java, using CoGenTex's RealPro (Lavoie and Rambow, 1997) for its text realization. Chapter 4 also evaluates CAPUT with the principles described in Chapter 3. Chapter 5 provides annotated examples of CAPUT's input and output.

Finally, Chapter 6 outlines the contributions of this thesis and suggests future directions of research.

Chapter 2

Related work

A variety of previous research projects have dealt with the issue of automatically generating presentations of quantitative data. Some work has gone into pure graphical presentations; some systems generate stand-alone textual summaries of the data; and others attempt to produce integrated graphics-text presentations. The following sections describe a number of such projects.

2.1 Graphical presentations

2.1.1 Manuals and guidelines

Several authors have created manuals to help presenters to choose the most appropriate graphical presentation technique in a variety of situations. For example, Stephen Kosslyn (1994), a Harvard professor of psychology, provides a classification of graphic design techniques motivated by principles of human visual perception and cognition. Gene Zelazny (1996) provides another set of guidelines for presentations, aimed at the business community.

Edward Tufte's books (Tufte, 1983; Tufte, 1990) provide a different perspective on graphic presentations. Tufte (1983) presents a showcase of interesting information graphics from their first use in the 18th century to the present day. He also provides a language for discussing graphics and a theory of data graphics which describes principles that authors should follow in creating graphic presentations of data. In his following book, Tufte (1990) extends this work into other graphical presentations—of geometry, of maps, and of train schedules, for example. Despite the theory of data graphics proposed in the first book, Tufte's books are, as he says, "celebration[s] of data graphics" (Tufte, 1983) rather than prescriptive manuals.

2.1.2 APT

Jock Mackinlay (1986) undertook one of the first projects to automatically produce effective graphical presentations of relational information. His APT system automatically designed two-dimensional static presentations of relational information. The core of the system is a precise definition of graphical languages that describe the syntactic and semantic properties of graphical presentations.

Various graphic design techniques are codified with criteria for expressiveness and effectiveness. Expressiveness criteria identify the techniques which that are capable of expressing the desired information; effectiveness criteria identify the most techniques that are most effective, in a particular situation, at exploiting the capabilities of the output medium and the human visual system. Designs are generated by a compositional algebra, which combines primitive graphical languages using operators to form complex presentations.

The input to APT consists of a database of application-specific information, such as properties of cars, and a request from the user to present this information in a certain way. A typical user input is the following (Mackinlay, 1986): "Present the Price and Mileage relations. The details about the set of Cars can be omitted". Given this input, the system produces a graphical design and an image rendered from the design. Figure 2.1 shows a graph produced by APT for this sample input. The specification that the details about the set of cars can be omitted makes it possible to use this presentation technique; if the car makes and models had to be presented as well, then this graph would not express all of the input.

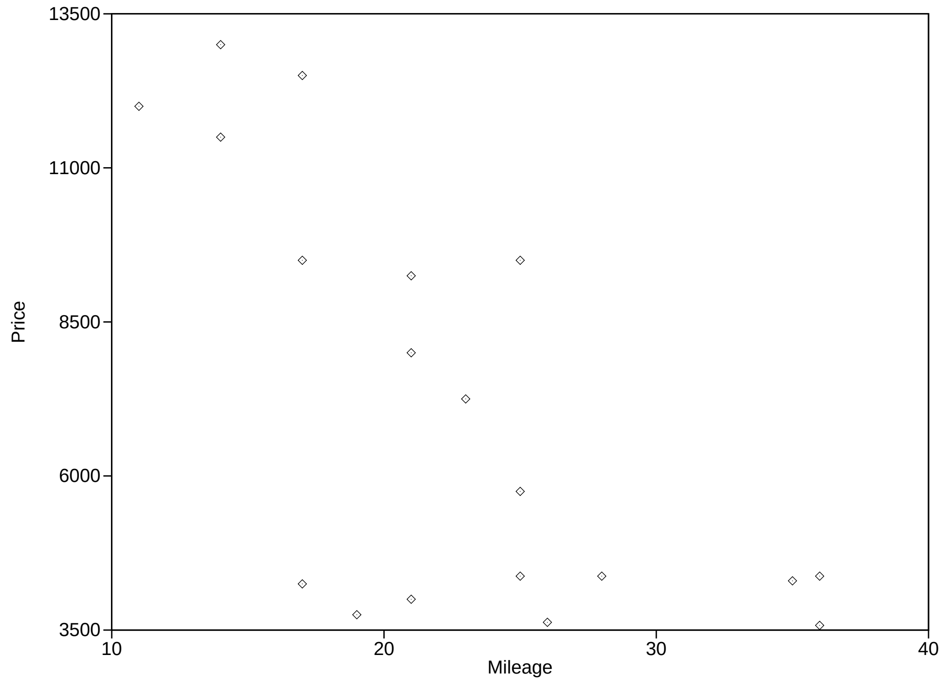


Figure 2.1: Sample APT output

2.1.3 Summary

Much of the previous work in this area consists of books containing guidelines for human presenters. These guidelines are useful aids in the development of an automated presentation system, but they do not provide an actual model which can easily be implemented.

One system which does implement an automated presentation tool is APT. Its codification of the expressiveness of various graphical presentation techniques is useful; however, neither APT nor the books of guidelines consider the issues of the text which is to accompany the generated graphic.

2.2 Stand-alone text

2.2.1 Ana

One of the earliest systems to produce automated summaries of quantitative data is Karen Kukich's Ana (Kukich, 1983), which generates textual summaries of stock-market data. Kukich gives the name "knowledge-based report generation" to the process followed by Ana; it has three basic tenets. First, it assumes that domain-specific semantic, linguistic, and rhetorical knowledge is required for a computer to produce intelligent and fluent text. Second, it assumes that production-system languages, such as those used to build expert systems, are well-suited to the task of representing and integrating the necessary language. Third, it assumes that "macro-level knowledge units" (i.e., phrases and clauses rather than words) provide an appropriate level of knowledge representation for generating summary reports.

The system has four components: a fact generator, a message generator, a discourse organizer, and a text generator. These act in series, with the output of one module serving as the input to the next. The fundamental knowledge constructs in the system are of two types: static knowledge structures (n-dimensional propositions) and dynamic knowledge structures (production rules).

The fact generator and message generator together extract the interesting points from the data, such as the fact that the market was "mixed". They use approximately 120 domain-specific inference rules created using a sample corpus to map the data into facts. Next, the discourse organizer orders the messages, groups them into paragraphs, and assigns a priority number to each message as a function of the topic and subtopic of a message. Ana has a default ordering, with exception rules that ensure that higher priority is given to messages with special significance (such as a record high). Finally, the text generator selects phrases from a lexicon extracted from the text of stock market reports and combines those which capture the meaning of the message and satisfy rhetorical constraints.

The output of Ana can be tailored syntactically by setting a variety of parameters to con-

01/12	CLOSE	30	INDUS	1083.61
01/12	330PM	30	INDUS	1089.40
01/12	3PM	30	INDUS	1093.44
01/12	230PM	30	INDUS	1100.07
01/12	2PM	30	INDUS	1095.38
01/12	130PM	30	INDUS	1095.75
01/12	1PM	30	INDUS	1095.84
01/12	1230PM	30	INDUS	1095.75
01/12	NOON	30	INDUS	1092.35
01/12	1130AM	30	INDUS	1089.40
01/12	11AM	30	INDUS	1085.08
01/12	1030AM	30	INDUS	1085.36
01/11	CLOSE	30	INDUS	1083.79

(a) Sample data

after climbing steadily through most of the morning, the stock market was pushed downhill late in the day. stock prices posted a small loss, with the indexes turning in a mixed showing yesterday in brisk trading.
the Dow Jones average of 30 industrials surrendered a 16.28 gain at 4pm and declined slightly, finishing the day at 1083.61, off 0.18 points.

(b) Generated text

Figure 2.2: Sample Ana input and output

trol the probability that specific syntactic rules are used during the text generation stage. For example, if the user prefers reports with few subordinate participial clauses, the corresponding parameter could be set to a low value. The same mechanism can also be used to tailor the content of the presentation, such as by focusing on specific stocks.

A sample of the input to Ana is shown in Figure 2.2(a). This data consists of Dow Jones stock quotes for a particular day in January 1983. An interpretation of this data generated by Ana is shown in Figure 2.2(b).

2.2.2 LFS

LFS (Iordanskaja et al., 1992) is an experimental system which generates bilingual (English and French) statistical reports on labour force statistics. The text planning process takes place as follows. First, a structure called a "conceptual frame tree schema" is instantiated with input

Seasonally Adjusted Estimates of Canadian Labour Force by Age and Sex (in 000s)

Month	Total (ALL)	Males	Fem	Total (15--24yrs)	Males	Fem	Total (25yrs+)	Males	Fem
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
11/1989	13600	7556	6044	2660	1408	1252	10940	6148	4792
10/1989	13538	7535	6003	2652	1399	1253	10886	6136	4750
09/1989	13528	7554	5974	2650	1407	1252	10878	6147	4731
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 2.3: Sample LFS raw input fragment (Boyd, 1999)

Overview: Estimates for November 1989 from Statistics Canada's Labour Force Survey show that the seasonally adjusted level of employment rose by 32000 and that the level of unemployment increased by 30000. The unemployment rate increased by 0.2 to 7.6.

(a) English

Aperçu: Les estimations tirées de l'enquête de Statistique Canada sur la population active pour novembre 1989 indiquent que le niveau désaisonnalisé de l'emploi a augmenté de 32000 et que le niveau de chômage a augmenté de 30000. Le taux de chômage a augmenté de 0.2 à 7.6.

(b) French

Figure 2.4: Sample LFS output

data (relational tables) to provide an initial characterization of the intended report content. This tree is then traversed and modified by a process which determines the detailed content of the final report.

The output of the text planning process is represented in a conceptual interlingua, which is then converted to a different semantic net for each output language; this permits the production of sentences which show deep differences between English and French. These nets are then realized in the respective target languages by a process based on Mel'čuk's Meaning-Text Theory (Mel'čuk, 1988). There are four successive levels of representation between the conceptual structures and the final texts: semantic nets, deep syntactic trees, shallow syntactic trees, and morphological strings.

A sample of the raw input to LFS is shown in Figure 2.3; Figure 2.4 shows a portion of the sample output presented by Iordanskaja et al. (1992).

2.2.3 FOG

One of the few commercial systems for generating textual descriptions from numeric data is FOG (Goldberg, Drieger and Kittredge, 1994), a system which produces routine and special-purpose forecasts directly from graphical weather depictions. FOG converts data to textual forecasts in three stages.

The first step is to extract specific data, such as the surface air temperature in a specific forecast area. This is done with the help of the Forecast Production Assistant (FPA), which helps the forecaster develop a time series of weather depiction charts which are to be the input to the system. FOG then uses a sampler program to extract the necessary information from each of a set of preassigned sample points.

Next, the data is processed to extract significant events, where the significance of a weather event is determined both by the needs of the meteorologist and by the other events occurring at the same time. An expert system designed to mimic a meteorologist extracts these significant events. Forecast areas are grouped by similarity in order to minimize the length of the text. Depending on the type of forecast, different groupings are used: for marine forecasts, the messages are ordered strictly by data salience (relative significance to the intended user), while for public forecasts, the messages are grouped by temporal order and then by salience.

The final stage is linguistic processing, which has two major stages: text planning and text realization. As in LFS, the text realization follows Mel'čuk's Meaning-Text Theory; for this example (taken from Goldberg and Driedger (1994)), we will consider only the text-planning stage. The table in Figure 2.5(a) shows some sample data on wind speed. The left-hand columns represent the data sampled from the FPA's graphical weather depictions. FOG then classifies the data into wind direction and speed and "time-merges" them, as shown in the right-hand columns. (When the sampled wind speed falls below 13 knots, it is classified simply as light; the direction is no longer considered significant.) The dots in the middle of the table represent data which is not relevant to this example.

Once the meaningful categories have been identified (as shown on the right-hand side of

	Sample Data		Concepts	
Time	Wind Direction	Wind Speed	Wind Direction	Wind Speed
6 a.m.	223	13	southwest	15-20
7 a.m.	235	17	⇓	⇓
9 a.m.	231	21	⇓	⇓
⋮	⋮	⋮	⋮	⋮
9 p.m.	280	12	(west)	light
10 p.m.	307	11	(northwest)	⇓
11 p.m.	182	8	(south)	⇓
Midnight	246	10	(southwest)	⇓

(a) Sample data

"Winds southwest 15 to 20 knots diminishing to light late this evening."

(b) Generated text

Figure 2.5: Sample FOG input and output

Figure 2.5(a)), the text planner must build concepts to describe transitions between the states. It then outputs an interlingua representation of the concepts, grouped into sentence-sized chunks. In the example, the two wind events are grouped together. The text generated for this example is shown in Figure 2.5(b).

2.2.4 GOSSIP

The goal of GOSSIP (Carcagno and Iordanskaja, 1993) is to inform a security officer about the operations performed by the users of a computer centre—for example, which files are used or deleted and at what time. It can produce verbal or graphical reports concerning the behaviour of the system or of particular users. GOSSIP comes from the same group of researchers that produced LFS (Section 2.2.2) and FOG (Section 2.2.3).

The input to GOSSIP is the audit trail produced by an operating system. A fragment of such an audit trail is shown in Figure 2.6; it shows a particular user, "jessie", logging in and running various programs shortly before 1:00 AM. In this form, the data is difficult to read and does not lend itself to immediate conclusions; it must therefore be summarized and interpreted.

```
ee(jessie, term32, success, [ ], login, [ ],
   nt, 00:48:08, 00:48:08, 00:00:00, 0.000, 2373131)
ee(jessie, term32, success, [ ], dir, [ ],
   nt, 00:48:23, 00:48:23, 00:00:00, 0.000, 2373131)
ee(jessie, term32, success, [ ], edit, [ ],
   nt, 00:48:44, 00:53:29, 00:04:44, 56.816, 2373131)
ee(jessie, term32, success, [ ], c, [ ],
   nt, 00:53:57, 00:55:25, 00:01:28, 17.621, 2373131)
```

Figure 2.6: Sample GOSSIP input

The system was operating for 6 hours 36 minutes and 57 seconds. Usage was particularly intense between 16:32:03 and 18:54:29 with idle time only 27 cycles during this period. Seven users worked on the system. Five of them used mostly compilers (C, Lisp, Fortran) and the Prolog interpreter. VLADIMIR and LEO read numerous files. VLADIMIR was interested in system priority tables. LEO listed many user files from his own group. He initiated large print jobs using these files. VLADIMIR failed to change access parameters for system files. No modifications to system files were noted.

Figure 2.7: Sample GOSSIP output

The process used to plan the text in GOSSIP is similar to that employed by LFS. The user requests a particular report, and a "topic tree" is activated that defines the topics typically addressed in that type of report. Initially, the topic tree is a collection of potential topics, organized hierarchically. The tree is then traversed by a procedure which triggers a method on each node to determine the content; the leaves are instantiated by looking for facts in the audit database, while other topics compute their content from other already instantiated topics. The text is also structured at this point: elementary messages which are compatible are grouped into more complex messages (e.g., if several users spent time editing files, those actions would be grouped together); messages which are not applicable to the particular situation are removed (e.g., a user reading her own files may not be of interest); and new messages which result from the content of existing messages are inserted (e.g., if a report of printing activity is to be included, then a list of the files printed may be necessary). Finally, a semantic representation of the individual messages is produced; this is then sent to the linguistic component for realization. As in LFS and FOG, the realization follows a process based on Meaning-Text Theory. An example of the output of GOSSIP is shown in Figure 2.7.

2.2.5 STREAK

Robin and McKeown (1996) describe STREAK, an experimental system to generate summaries of the events of basketball games in the style of the lead sentences of newspaper sports reports.

Several characteristics of the corpus of basketball game summaries influenced the approach taken. The complexity of the sentences in the corpus was great, with four to twelve simple facts conveyed in a single sentence. While some concepts consistently appear in fixed locations across reports, others appear wherever the form of the surrounding text allows. To be able to add such "floating facts" wherever necessary, the facts must be expressible in a variety of linguistic forms. Summaries may contain historical background facts to highlight the significance of new reported facts. Summaries must also be concise, conveying as much information as possible in a limited space.

To take into account these characteristics, STREAK uses a model of text generation which relies heavily on revision: first a bare-bones summary is generated, and then additional information is added opportunistically until there is no space left. A set of revision rules specify the various ways a draft can be modified to accommodate a new piece of information; these rules were derived empirically from a set of human-written sports summaries. The revision rules act directly on the text plans.

The input to STREAK consists of two semantic nets, one representing the facts which must be conveyed and one for the "floating facts" (optional additional information). Figure 2.8 shows some sample output from STREAK, adapted from Robin and McKeown (1996). The first draft of the summary of a particular game is shown in Figure 2.8(a). This first draft contains only minimal information about the game: the location, main individual statistic (Barkley's point total), date, and game result. A series of complementary facts is added one at a time, resulting in some of the words of the initial draft being deleted, displaced, or transformed. After this process is completed, the sentence shown at the in Figure 2.8(b) is the final result.

The use of a corpus allowed STREAK to be empirically evaluated by comparing the generated texts to those already in the corpus. Two aspects of robustness were evaluated: coverage

"Dallas, TX—Charles Barkley scored 42 points Sunday as the Phoenix Suns defeated the Dallas Mavericks 123—97."	"Dallas, TX—Charles Barkley tied a season high with 42 points and Danny Ainge came off the bench to add 21 Sunday as the Phoenix Suns handed the Dallas Mavericks their league worst 13th straight home defeat 123—97."
(a) Initial draft	(b) Final sentence

Figure 2.8: Sample STREAK output

and extensibility. To evaluate coverage, the generator was implemented on one year's worth of basketball summaries, and then the sentences from a different one-year sample that it could produce were counted. Extensibility was measured by counting how many additional knowledge structures were necessary to completely cover an additional year. The revision model increased the overall realization coverage by 41 percentage points over a one-pass model, and the extensibility by 14.6%.

The revision rules were also evaluated for cross-domain portability by examining a corpus of stock-market reports (such as those used in Ana). It was found that about 70 per cent of the revision rules also applied to this new domain, although Robin and McKeown (1996) do not specify how many sentences in the corpus used rules other than those gathered from the basketball sentences.

2.2.6 TREND

A more recent system is Sarah Boyd's TREND (Boyd, 1999). TREND detects and summarizes short- and long-term trends in time-series data; the initial domain is currency exchange data. It uses wavelets, a signal-processing technique originating in mathematics, to detect the trends.

The input to TREND consists of an "annual currency file", which consists of a number of lines each containing the date and the daily currency value. As well, the system user can use a graphical interface to specify the settings of parameters that control the sort of information extracted from the input data—the length of the trends to detect (short, long, or both), and whether high and low volatility periods are included.

The input and the settings of the user parameters are fed to a content determination module, which performs the following three steps. First, the basic visual features of the data are identified (trends, low-volatility intervals, and high-volatility intervals); it is here that the wavelet analysis is used. The settings of the user parameters determine which features are extracted from a given data set. Once the visual features are extracted, any which overlap in time are merged, using heuristics to select which of these features should be described. Finally, the non-overlapping features are aggregated—for example, an increase immediately followed by a decrease is identified as a peak. The output of this process is an Aggregated Visual Feature Matrix (AVFM).

Next, the AVFM is fed into a module which performs the tasks of document structuring, aggregation, and lexicalization. The schema begins by describing the overall yearly trend, and then describes the other visual features in temporal order. The visual features are grouped into sentences by an algorithm which puts features into the same sentence until an aggregated feature is encountered, at which point a new sentence is started. Next, phrasal templates are selected to describe each of the features; these templates take the form of functional descriptions for FUF/SURGE (Elhadad and Robin, 1996). Finally, the templates are sent to FUF/SURGE for realization.

The final output from TREND consists of the multi-sentence summary of the selected visual characteristics, as well as a graph of the data annotated to indicate the characteristics described—for example, red arrows indicate long-term trends.

A sample of the input to TREND is shown in Figure 2.9; this data represents the value of the Australian dollar measured against the U.S. dollar during January 1997. Sample text generated by TREND on the data for the full year is shown in Figure 2.10. For this output, the user requested that both short- and long-term trends be described, and that areas of low and very high volatility should be included.

0.7895,	02,	01,	97
0.7914,	03,	01,	97
0.7882,	06,	01,	97
0.7891,	07,	01,	97
0.7796,	08,	01,	97
0.7780,	09,	01,	97
0.7818,	10,	01,	97
0.7783,	13,	01,	97
0.7810,	14,	01,	97
0.7767,	15,	01,	97
0.7791,	16,	01,	97
0.7800,	17,	01,	97
0.7774,	21,	01,	97
0.7756,	22,	01,	97
0.7716,	23,	01,	97
0.7720,	24,	01,	97
0.7722,	27,	01,	97
0.7690,	28,	01,	97
0.7700,	29,	01,	97
0.7657,	30,	01,	97
...			

Figure 2.9: Sample TREND input

During 1997, the currency fell 17.47 percent to finish the year at 0.651. It remained mainly unchanged between the 20th of February and the 30th of May and decreased considerably between the 30th of May and the 7th of July before staying mainly unchanged until the 10th of September. It fell dramatically between the 10th of September and the 31st of December.

Figure 2.10: Sample TREND output

2.2.7 Summary

The projects described in this section take a variety of approaches to producing text which appropriately describes the input domain. All of them are tailored to a specific domain—the stock market, labour force statistics, currency prices, or basketball scores; STREAK does consider the possibility of being used in other domains as well, though. TREND and GOSSIP also produce graphics, but the text is designed to stand on its own.

The various systems use different methods of actually producing the text. Some use a phrasal lexicon or some other form of textual template to produce the text—Ana falls into this category, and TREND partly does as well. Other systems use full-fledged linguistic realizers to produce the text. LFS, FOG, and GOSSIP use a similar one, an ancestor of RealPro; STREAK uses FUF/SURGE, and TREND's phrasal templates are also realized using this system.

Notice that text which is generated to stand on its own differs from text whose aim is to accompany a graphic (see Section 3.1.2 for more on this issue). For example, often it is better to mention only some of the message in the text, and to allow the audience to infer the rest from inspection of the graphic. The techniques used in the projects in this section to select data to be presented and to produce appropriate textual presentations from it can be applied to integrated text and graphics systems; however, text which is generated to stand on its own will often be too detailed or contain redundant information when paired with a graphic. Such other factors should also be taken into account if integrated presentations are to be produced.

2.3 Integrated graphics and text

2.3.1 SAGE

The SAGE project at Carnegie-Mellon University concentrates on the automated generation visualizations of complex data. The SAGE system is an expert system specialized in graphic design. Most of the work on SAGE itself has gone into the automated generation of graphical data visualizations; however, there have been several related projects which also aim to generate text. One, AutoBrief, is described in Section 2.3.3. Another project, described in Mittal et al. (1998), presents techniques for producing captions for SAGE-generated graphics; however, this research has concentrated on producing captions that explain the graphical techniques used in producing the graphics, rather than pointing out aspects of interest in the data being displayed on them.

Roth, Mattis and Mesnard (1991) describe an application which uses SAGE to produce coordinated text-graphics presentations in response to users' questions. The domain of this application is project management; managers often use project modelling systems to evaluate a project under a number of hypothetical conditions, and will often need to ask the system questions about the value of some variables under different conditions. Examples of typical questions include:

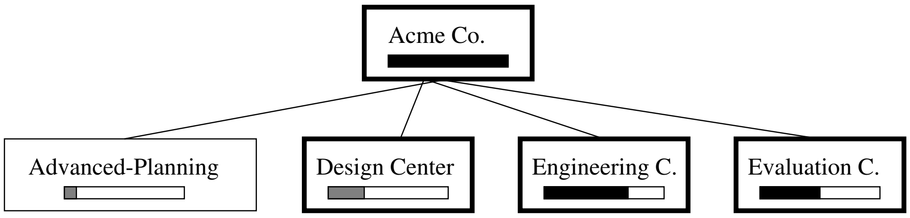
- Why did Bill Smith's activities cost more than expected in the estimate?
- Why is the end date of the design approval activity later in this week's schedule, as compared to last week's?

To allow these sorts of questions, the underlying data is represented in three models: a financial spreadsheet model, an "alternative additive model" (which represents the organizational structure of the company), and a scheduling model (which represents the start and end time of each activity in the project). These models are linked together to provide an integrated representation of the data; for example, links are made from the departments in the organizational hierarchy to the activities which those departments may perform.

The process of producing a response to a user's question proceeds as follows. First, the content is selected from the underlying data in a two-stage process. All of the data in the model which could possibly be relevant to the query is first extracted. That data is then analyzed to derive further facts from it; these facts can also form the content of the presentation.

Next, a discourse component selects the assertions which will appear in the output and orders them appropriately. Different components are used depending on the type of question asked by the user; for example, *cause identification* consists of a statement of the main cause-effect relationship, any counteracting or reinforcing relationships, and the main dependency that the causal relations are based on. The output of the discourse processor is an ordered sequence of instantiated rhetorical predicates. These are then input to an external text realizer, which smooths the resulting text by adding sentence transitions, making definite reference to entities, and performing some forms of ellipsis.

So far, only the textual realization component has been described. However, some assertions are better made in graphics than in natural language: for example, organizational relations or the precedence of activities. As well, if facts are presented in natural language, then the user must pay full attention to each fact; there is no way to present many facts in such a way that the



> “Why did Acme-Co costs change from estimate-1 to estimate-2?”
(1) The increase in Acme-Co costs was due to increases in Engineering Center and Evaluation Center, as shown by the gauges in Figure A.
(2) These increases were partially counteracted by a decrease in Design Center.

Figure 2.11: Sample SAGE output

user can focus only on the interesting ones. For these reasons, SAGE also produces graphical displays.

Several problems can arise if the text and graphics are generated independently. The sequential structure of information in the text and the graphical structure of that information in a picture might not correspond; the text cannot refer to the graphical representations of the domain objects that are being discussed; and the two presentations may contain redundant information. The two generators therefore communicate with each other to increase the coordination in the final presentation. The two parts of the presentation are produced and coordinated as follows. First, some discourse processing is done to create an idea outline that partitions the relevant content into a logical sequence and possibly combines information from different discourse segments as a single merged set. Next, the graphics generator produces its portion of the output; finally, the text is produced, making explicit reference to the components of the graphical presentation and omitting propositions that are already expressed in the graphics.

An example of a portion of a coordinated text and graphics display produced by this system is shown in Figure 2.11. The graphic and text are a portion of a more complete sample presented by Roth, Mattis and Mesnard (1991); the full example from the paper includes several more levels in the hierarchy of departments, which are also referenced in the full caption.

2.3.2 PostGraphe and SelTex

PostGraphe

PostGraphe (Fasciano, 1996) is a system that generates integrated graphics and text presentations from statistical data. The input to PostGraphe is in the form of a Prolog term specifying the characteristics of the data to be presented, the goals of the user, and the data itself. A sample of the input format[1] is shown in Figure 2.12; this example is taken from Corio (1999). The lines that specify candidates and non-candidates for keys control which of the variables may be used as a relational key in preparing the presentation. The system generates a report containing appropriate graphics and text that is based on the contents of this input file.

PostGraphe always generates a text-graphics pair for every message. The generation of both components of the presentation is done in the same way, as follows. A planning algorithm is used to generate a schema for each group of compatible inter-variable or intra-variable goals. This schema is used for both graphics and text. To trim down the search space of potentially compatible groups of goals, heuristics are used; as well, the user may manually limit the search space by building sets of related goals in the input.

To choose the appropriate schema, a table is used which associates each possible user goal with the schemata that can express it and a weight indicating how efficient each schema is at presenting that goal. All of the knowledge used in schema selection is encoded in these weights, in an approach similar to neural nets.

Figure 2.13, adapted from Corio (1999), shows the result of running PostGraphe on the input file in Figure 2.12. The table at the top of the figure is the result of the intention of the first section, *presentation*; the line graph in the middle results from the first intention in the second section, *comparison*; the text at the bottom results from the third intention, *evolution*. The text in this figure is the only sort that PostGraphe itself is capable of producing.

[1]The translation of all extracts from Fasciano (1996) and Corio (1999) from the original French is my own.

```
data(% names of variables
     [year,company,profits],
     % types of variables
     [year/[symbolic],
      label,
      dollar/[plural(profit)]],
     % candidates for keys
     [year,company],
     % non-candidates for keys
     [profits],
     % author's intentions
     [% section 1
      [presentation(year),
       presentation(company),
       presentation(profits)],
      % section 2
      [comparison([profits],[company]),
       evolution(profits,year)]],
     % the raw data
     [[1987,'A',30],
      [1988,'A',35],
      [1989,'A',40],
      [1990,'A',35],
      [1987,'B',160],
      [1988,'B',165],
      [1989,'B',140],
      [1990,'B',155],
      [1987,'C',50],
      [1988,'C',55],
      [1989,'C',60],
      [1990,'C',95]]).
```

Figure 2.12: Sample PostGraphe input

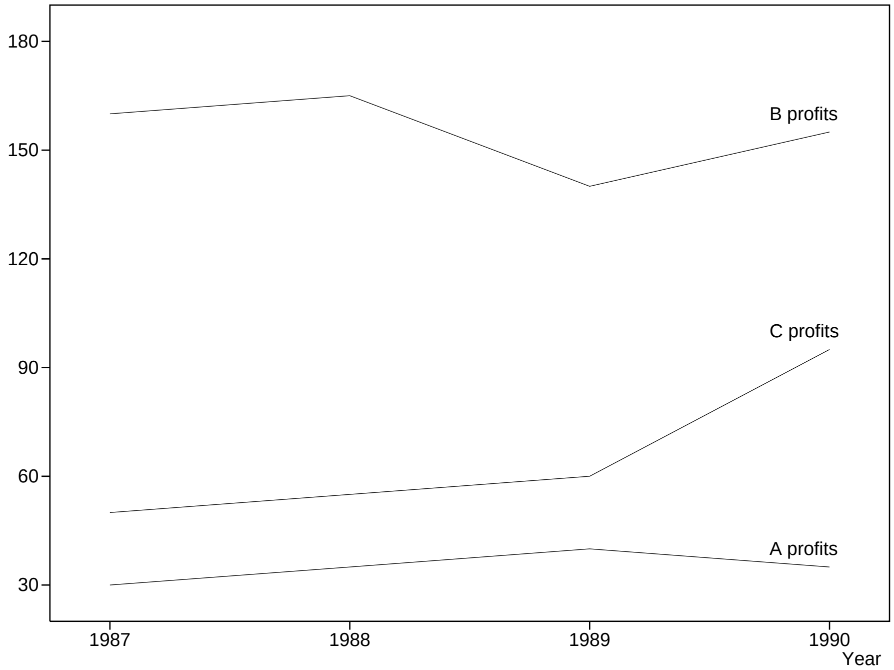
SelTex

Marc Corio has recently completed a thesis (Corio, 1999) in which he implemented a module, SelTex, to improve PostGraphe's somewhat limited text-generation capabilities.

Corio performed an extensive corpus analysis of French text-graphics pairs, mostly from Statistics Canada publications, and classified the types of text into 55 different codes. One example of such a code is BAS1, which refers to a text that describes "[t]he lowest column, the shortest bar or the smallest sector of the pie".

He then grouped these codes by the intention with which they were most frequently associated. For example, 50 per cent of the captions which fulfilled the intention of *presentation* were

year company	1987 profits	1988 profits	1989 profits	1990 profits
A	30	35	40	35
B	160	165	140	155
C	50	55	60	95



From 1987 to 1989 the profits of company A increased from \$30 to \$40. Up to 1990 they decreased from \$40 to \$35.
From 1987 to 1988 B's profits increased from \$160 to \$165. During 1 year they decreased by \$25. Up to 1990 they increased from \$140 to \$155.
From 1987 to 1990 C's profits increased from \$50 to \$95.

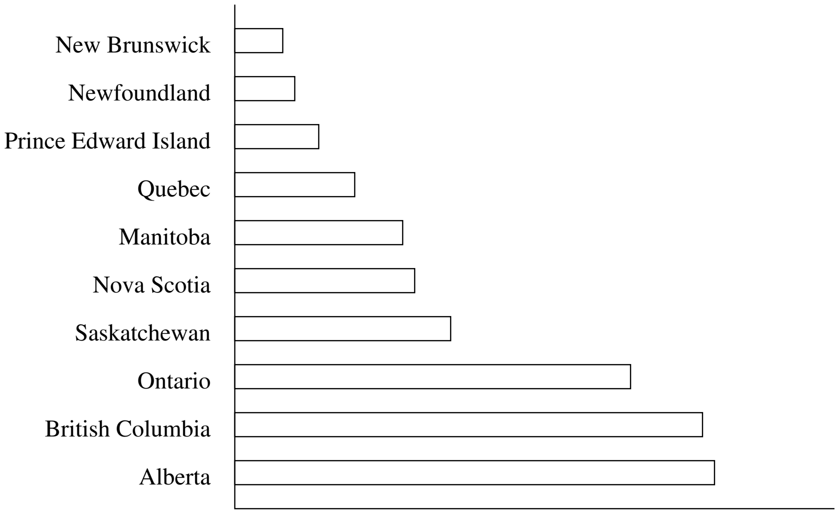
Figure 2.13: Sample PostGraphe output

of type TITRE2, "Generic title including the description of two or more data items".

Next, Corio produced a set of selection techniques to determine what sort of caption to produce given the nature of the data and the basic intention selected by the user. These selection techniques take the form of rules such as the following (for use with the *comparison* intention):

- Mention the highest data point if its value is at least 10% larger than the second and if the number of points is greater than 2.

These rules were implemented in Prolog, as was the rest of PostGraphe. Some enhancements were made to the input format of PostGraphe to enable users to specify various information



Alberta, British Columbia and Ontario have a higher rate of households with a computer, while Quebec placed seventh with 24.0%.

Figure 2.14: Sample PostGraphe output (with SelTex)

useful to SelTex.

SelTex contains 15 new textual schemata which are associated with the possible classification codes. To select the appropriate schema, the following steps are followed. After the base PostGraphe system is used to generate a graph (as described by Fasciano (1996)), SelTex is called. The first step is to verify whether SelTex has a schema which can satisfy the desired intentions. Once the schema has been obtained, the next step is to find the appropriate predicate to execute in order to generate text using the schema. If all of the necessary conditions are satisfied to generate text using the schema, a "message" is generated. Finally, this message is passed to the text generator, which in this case is a Prolog rewriting of FRANA (a French translation of Kukich's Ana (Section 2.2.1)).

The format of the input to SelTex is the same as that to PostGraphe, with a few additions to control the text generation. Figure 2.14 shows an example of the output of SelTex, adapted from Corio (1999). The main intention in this case was comparison of computer use between provinces, with a secondary goal to concentrate specifically on the value for Quebec.

2.3.3 AutoBrief

AutoBrief (Kerpedjiev et al., 1997) is a system whose overall aim is to produce coordinated multimedia explanations of large and complex datasets. The output it produces is designed to help analysts in dealing with such datasets and in presenting the results of their analyses to others.

AutoBrief combines features of two complementary previous approaches to automatic presentation design: hierarchical planning to achieve communicative goals, and task-based graphical design. The interface between these two components is a domain- and medium-independent layer of communicative goals and actions.

The main test domain for AutoBrief, transportation scheduling, is described by Kerpedjiev et al. (1997). In this domain, transportation analysts and planners use systems which produce a number of schedules for moving commodities around. These schedules are analyzed for lateness or bottlenecks, and the planners may then suggest workarounds. The goal of AutoBrief is to help the analysts in performing their task. AutoBrief creates multimedia summaries of schedules, containing graphs, tables, and some textual information about the capabilities available and the shortfalls.

AutoBrief has access to a knowledge base of information about the particular domain in which it is working. The input consists of the a set of domain-specific communicative goals, such as *know-shortfalls* (ensure that the user knows the value of the *shortfalls* attribute). These goals are refined into domain- and medium-independent subgoals, which are in turn achieved by domain- and medium-independent abstract actions. These abstract actions are decomposed into medium-specific actions, which the medium-specific generators then use to produce the final presentation. AutoBrief uses SAGE (Section 2.3.1) to produce its graphics and FUF/SURGE (Elhadad and Robin, 1996) as its text realizer.

As an example of the process, the goal *know-shortfalls* could be refined into the subgoal *know-attribute*; this subgoal is realized by the action *assert*, which can then be decomposed into the graphical action *enable-lookup*. In other words, if the final presentation includes a graphic on which the user can effectively look up the value of the *shortfalls* attribute, then the goal has been achieved.

Kerpedjiev et al. (1997) describe the use of AutoBrief in the transportation scheduling domain; a series of papers from 1998 provide more detail about the inner workings of AutoBrief and other possible applications for the system, using newspaper readership numbers as a source of examples.

The content language which is used to represent the medium-independent subgoals and the content of the medium-independent actions is described by Green et al. (1998a). The language represents what is to be asserted rather than the type of communicative acts to perform or the attitudes which the acts are intended to achieve. This language needs to be able to represent complex descriptions of quantitative database attributes and to represent them compositionally, with possible subtle differences in intention for the same data. It also must be medium-independent, while still providing the necessary information for both medium-specific generators.

Green et al. (1998b) describe how certain types of arguments that can be represented visually in information graphics can be generated from an underlying medium-independent representation. Here, an "argument" is a presentation which, given knowledge of the user's current beliefs, aims to convince the user to accept a particular belief. This goal is then decomposed as described above into medium-dependent communicative acts to produce the presentation.

Figure 2.15, adapted from Green et al. (1998b), shows a possible medium-independent discourse plan for the realization of a particular argument. The goal is to convince the user that the Post-Gazette newspaper has more readers than the total number of readers of all other newspapers that are distributed in a particular region (fact T in the discourse plan). The user currently knows that the New York Times has more readers than the Wall Street Journal (Q), and mistakenly believes that this means that the New York Times has the most readers in the region (R). This plan might be realized in text as in Figure 2.16(a), or by the graphic and caption shown in Figure 2.16(b). This output was presented by Green et al. (1998b) as examples of possible realizations of the discourse plan; it was not actually produced by AutoBrief.

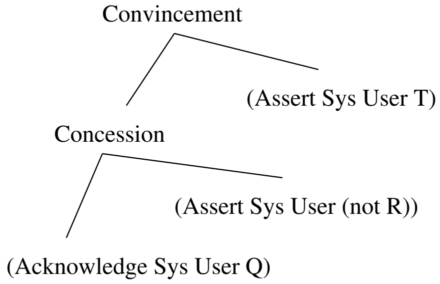
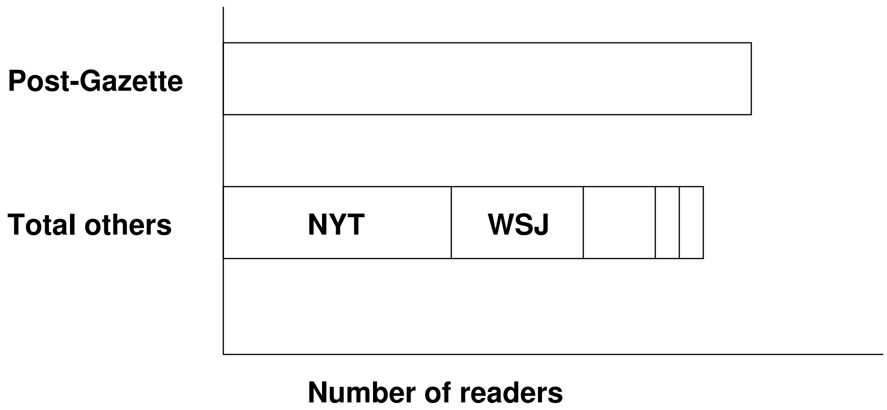


Figure 2.15: Sample AutoBrief discourse plan

Although the New York Times is read by more people in Western PA than the Wall Street Journal, the New York Times does not have the highest number of readers in the region. The Post-Gazette has more readers than the total number of readers of all other newspapers in the five-county Western PA region.

(a) Text



The Post-Gazette has more readers than the total number of all other newspapers in the five-county Western PA region.

(b) Text and graphics

Figure 2.16: Possible realizations of AutoBrief discourse plan

2.3.4 Summary

The systems in this section employ various techniques to generate text-graphics presentations of their data. The application of SAGE described here in detail is tied to a particular domain (project management); PostGraphe and AutoBrief aim to be domain-independent.

The graphics and text are produced and coordinated in different ways. SAGE runs its text and graphics generators concurrently so that they can communicate with each other to produce coordinated output; AutoBrief decomposes its domain-specific goals into communicative acts for each of its two generators, which then act independently; PostGraphe chooses its textual and graphical schemata separately, using the same input to guide both choices.

Of the systems described here, only SAGE considers the issue of medium allocation—that is, determining which output medium is best suited to generating each component of the message. In the other multimedia systems, messages are realized in whichever medium is capable of expressing them, without any consideration for coordination or redundancy.

Chapter 3

Principles of caption generation

3.1 Introduction

3.1.1 What is a caption?

Caption generation is the task of generating text designed to accompany an information graphic. Such text does not necessarily have to appear as a caption in the final presentation (i.e., above, below, or beside the graphic). It could also appear in an article accompanying the graphic, or could even be spoken in a presentation. However, for convenience, such text will be referred to in this thesis as a "caption", whatever position it takes in the final presentation.

A caption can serve two purposes: it can point out the relevant or interesting aspects of the data presented in the graph, and it can explain the meaning of the various graphical techniques used to produce the image (Kosslyn, 1994). This thesis focuses on text of the former type, which describes the data in the graph rather than features of the graph itself. A system which generates the latter sort of text is described by Mittal et al. (1998); however, producing such captions requires an intimate knowledge of the system producing the graphic which is being explained, and the techniques useful with one particular system might be completely useless in any other situation. Producing captions which describe the content of the graph, rather than the form, is a more general question.

3.1.2 Captions vs. stand-alone text

It is possible to produce text which summarizes quantitative information itself, without intending it to accompany a graphic of any sort; many of the systems described in Chapter 2 are of this type. However, the content of such stand-alone text is very different from text which is designed from the outset to go along with a graphical presentation. In particular, text in a caption is likely to mention only the general message which the author wants the audience to understand and to omit many of the specific details that can be read directly from the graph.

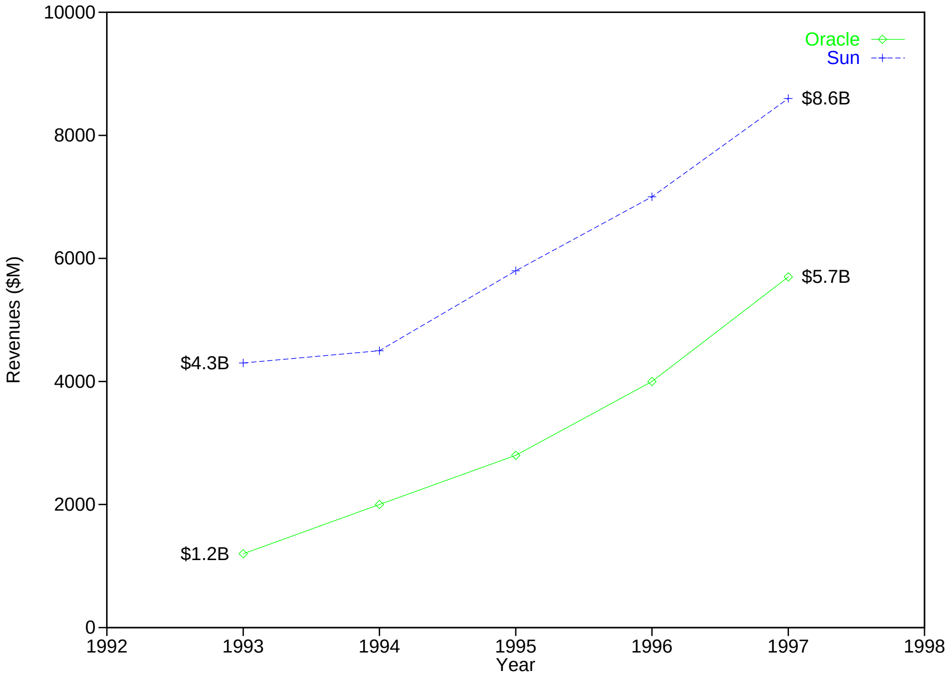
This point is illustrated by Kerpedjiev et al. (1998), which describes the process of manually transforming a purely textual summary of some quantitative data into an integrated graphics-text presentation. The original text, a portion of Bill Gates's U.S. Senate testimony from March 1998, contains several quantitative assertions; one such assertion is shown in Figure 3.1(a). Notice that the specific revenue increases of the two companies are spelled out explicitly in the text, as there is no other way in this medium of providing the examples needed to support his point. Messages of this type are often better presented with the use of graphics. Figure 3.1(b) shows a possible multimedia presentation, adapted from Kerpedjiev et al. (1998) of the same portion of the Gates testimony. In this case, the text contains only the main point which is being made (that the revenues of many of the companies in question have soared), while the particular data values are shown only on the graph. This multimedia approach exploits the capabilities of each medium produce a presentation which expresses the argument more efficiently than in the original pure text version.

The focus of this thesis is the generation of text designed to *accompany* graphics, rather than to stand alone. This means that, in contrast to the pure text systems described in Section 2.2, issues concerning the allocation of message components among the available media and the production of presentations coordinated across the media must be considered.

"[R]evenues from [sic] many of these companies have soared in recent years. (For example, Oracle's revenues rose from $1.2 billion in 1993 to $5.7 billion in 1997; over the same period, Sun's revenues rose from $4.3 billion to $8.6 billion.)"

(a) Original text

Revenues for many of these companies have soared in recent years.



(b) Text and graphics presentation

Figure 3.1: Gates testimony, in text and text-graphics presentations (Kerpedjiev et al., 1998)

3.2 Sources of information

The tasks and principles listed in the remainder of this chapter are derived from a variety of sources. The list of generation tasks comes from Reiter and Dale (1997), who present an outline of of a generalized text-generation architecture; this structure provides a useful framework for talking about the particular necessities of caption generation and its similarities to and differences from text generation as a whole.

An informal study of a number of information graphics and accompanying text from a variety of newspapers (*The Globe and Mail*, *USA Today*), mutual fund reports, and other sources provided data to help guide the creation of these principles. The formal corpus study of French

Province	**1990**	**1998**
Quebec	1400	1450
Alberta	1500	1400
British Columbia	1610	1900
Ontario	1700	1700
Maritimes	1575	1640

Figure 3.2: Sample data (healthcare spending)

text-graphics pairs in Marc Corio's thesis (Corio, 1999) was also an excellent source of data.

The techniques used in the related research projects described in Chapter 2 were another source of information about and examples of automatically producing presentations from quantitative data.

3.3 Tasks in caption generation

The goal of this thesis is to describe how to generate captions—that is, short pieces of text which are associated with a graphic presentation of some quantitative data. Although the process of caption generation follows the same steps as that of text generation as a whole, it differs in some ways.

Reiter and Dale (1997) describe an architecture for text generation which consists of six basic tasks: content determination, discourse planning, sentence aggregation, lexicalization, referring expression generation, and linguistic realization. The following sections outline how each of these tasks should be addressed in a caption-generation system.

Throughout the section, the data shown in Figure 3.2 will be used as an illustration. The data represents fictional spending on healthcare in a number of Canadian provinces in two different years.

3.3.1 Content determination

Content determination is "the process of deciding what information should be communicated in the text" (Reiter and Dale, 1997). For the purpose of this thesis, content determination will be defined as the process of selecting the relevant information from each *individual* dataset in the input. Combining these individual assertions into a single- or multi-sentence caption is the task of discourse planning and sentence aggregation; these tasks are described in subsequent sections.

The content selected to appear in a caption could include, among many other data features, the following: the value of a variable at the start, finish, or some other specific point, the overall or percentage increase or decrease, or the individual changes in the direction of a trend throughout a dataset. In general, the caption should contain information which helps the audience of a multimedia presentation to get the "right" impression from a graphic which forms part of that presentation. Several factors influence the content selected from the input: the type of graphic it is to accompany, the domain from which the underlying data is drawn, the features of the data itself, and the needs of the system user.

It is crucial, when choosing the information to include in a caption, to keep in mind that the final presentation will also include a graphical presentation of the data. It should complement the graphic it is to accompany without simply enumerating all of the data points; rather, it should *interpret* the graphic by emphasizing a particular aspect or aspects of the data. Specific features should be included only if there is a reason for so doing: for example, if the user has explicitly requested that the caption concentrate on those features, or if they are in some way unusual or interesting for the domain. Also, the message expressed by the text and by the graphic should be the same, or at least compatible, since a presentation with a graphic and text that are sending different messages will not be well understood.

The domain from which the data is drawn affects what sorts of features should be presented. For example, in an article comparing the changes in healthcare funding during the past decade using the sample data in Figure 3.2, it may be more interesting that the final value for Alberta was lower than its initial value; in an article comparing current funding across the provinces, it might be more interesting that British Columbia currently spends the most.

The features of the data itself also have an influence on the content which appears in the caption. Trivially, the values of the variables themselves will determine whether a caption should talk about an increase or a decrease. Less trivial choices can also be influenced by the data: for example, if the figures show an extremely large increase or decrease, such as that of British Columbia, then that feature may appear in the caption even if none of the other factors indicate that it should.

Finally, and most importantly, the needs of the user of the system should be taken into consideration when choosing the data to include. If there is a particular point that the user wants to drive home by using a graphic, then the caption should support that point, whether or not the other factors lead to its inclusion. For example, the user input could specify that the caption should concentrate on the value for the Maritimes, even if that variable does not have any other particularly interesting characteristics.

It should be possible to extend a caption-generation system to incorporate new rules for choosing content, so that appropriate text can be generated for a variety of purposes and data. These rules can be as general or specific as required for the particular situation.

Existing systems perform the task of content determination in various ways. In STREAK (Section 2.2.5), for example, the input consists of two semantic nets, one representing the facts which must be conveyed and one the "floating facts" which will be inserted if space permits; the content determination is thus mainly performed before the system even gets the data to present, with some determination taking place as floating facts are selected for inclusion.

Several systems depend on the particular domain for which they were written. For example, Ana (Section 2.2.1) uses approximately 120 inference rules derived from a corpus to extract "interesting messages" from the plain facts of stock market data. In FOG (Section 2.2.3), the process of choosing "significant events" is performed by an expert system which aims to mimic the choices made by a meteorologist. Such domain-dependent systems could be made to work in other domains, but it would require rewriting a number of rules (such as the 120 used in Ana) to adapt to the new domain. The content selected by SAGE (Section 2.3.1) and GOSSIP (Section 2.2.4) consists of two types of information: messages extracted directly from the raw data, and inferences made about the data. The process of making these inferences requires the system to have knowledge of the sorts of inferences to make.

Other systems use domain-independent methods of determining content. TREND (Section 2.2.6), for example, uses mathematical techniques from signal processing to detect trends in time-series data such as currency exchange information. In PostGraphe (Section 2.3.2), the textual schemata are weighted according to how effective they are at expressing a particular intention; the selected schemata then determine which messages are extracted from the data.

Of the integrated text-graphics systems, only SAGE specifically addresses the matter of avoiding redundancy between the text and the generated graphic; however, there is no medium allocation step before the presentation is produced.

3.3.2 Discourse planning

The goal of discourse planning is to structure the messages produced by the content determination process into a coherent text. In text generation as a whole, discourse planning often consists of organizing whole sentences or paragraphs. However, captions most often consist of a single sentence; in this case, therefore, the task of the discourse planner is to structure the messages within a sentence. As well, there may be length restrictions on a caption; if information must be deleted for space reasons, then the discourse planner should ensure that the most relevant information still appears in the final caption.

A simplistic form of structuring is to sort the messages by the magnitude of the value extracted; in the example data, this would put Ontario first and Quebec last. Other possible orderings include putting the most relevant message first (as determined by the user or by the system, making use of data features), putting messages of the same type next to each other, or placing messages which lend support after the message which they are supporting.

However, discourse planning does not consist just of putting the extracted messages into a linear order; it can also involve grouping messages by similarity or even removing some of the initial messages from the final content. For example, the messages in the example data might be structured by mentioning only those with extreme values, such as the highest value (British Columbia) or the sharpest decrease (Alberta). In such cases, the result is that some of the messages are dropped from the final sentence plan entirely. Discourse planning can even take the form of computing an average or total value and dropping *all* of the original component values from the sentence plan.

Whichever structuring technique a caption-generation system uses, it must have access to sufficient information about the individual messages in order to make appropriate decisions about the best order to put them into. In other words, the output of the content determination step must contain enough information about the messages it selects to enable the discourse planner to organize them properly.

The same set of messages can be structured in potentially many different ways, depending on the particular application. The discourse planning technique should be selected following the same criteria which drove the content determination stage: namely, the type of graphic it is to accompany, the domain, the features of the data, and the user's requirements.

In some cases, the line between discourse planning and content determination can blur. For example, in a caption which describes the correlation between the values of two variables, the description of the correlation could be viewed either as the content itself or as a structuring of the two individual messages in the discourse plan. These tasks may even be performed at the same time by the same component of the generation process.

As with the content determination rules, the rules of discourse planning can be as general or as specific as the particular situation warrants. Simple rules, such as sorting by value, should certainly be included. Corio (1999) gives a number of more complex heuristics derived from a corpus analysis; one such rule is the following:

- Mention the highest data point if its value is at least 10% larger than the second and if the number of points is greater than 2.

A full caption-generation system should be capable of implementing rules at least as complex as this. The rules used in a particular domain are best derived from a corpus of texts in the domain and/or through consultation with human domain experts.

Discourse planning is not addressed at all in many of the existing systems; some systems that do address it, however, are SelTex, Ana, FOG, GOSSIP, LFS, and STREAK. In SelTex (Section 2.3.2), rules such as the one described above actually form part of the content determination; there is no separate discourse planning step.

Ana's (Section 2.2.1) third module performs what Kukich calls the "uncomplicated task" of grouping messages into paragraphs, ordering messages within paragraphs, and assigning a priority number to each message. The priorities are assigned as a function of the topic and subtopic of a message. The system has a default ordering built in, with some exception rules that ensure that especially significant messages (such as an indicator hitting a record high) get a higher priority.

FOG (Section 2.2.3) uses two different techniques of ordering, depending on the type of forecast which is to be produced. For marine forecasts, data salience is used to order the various messages, where the salience of a piece of data is its relative significance to the intended user. For public forecasts, the messages are first grouped by temporal order, and then by salience within each temporal grouping.

In GOSSIP (Section 2.2.4), the following discourse planning technique is employed. First, a tree is created which represents the individual messages which can be expressed. This tree is then modified to combine messages which are compatible, to remove messages which are not applicable, and to insert new messages which result from the content of existing ones. The tree is then traversed in a top-down, depth-first manner to produce its texts. LFS (Section 2.2.2) also uses this basic technique.

Discourse planning in STREAK (Section 2.2.5) is somewhat different, due to its revision-based architecture. Information is added to the bare-bones initial schema on the fly to support or elaborate on the initial facts. It is the revision rules, which specify where and how such additional content is to be inserted, that act as a discourse planner in determining the structure of the content in the final output. Alone among the previous systems, STREAK incorporates length restrictions on its generated text, adding only as many floating facts as will fit into the allotted space.

3.3.3 Sentence aggregation

Sentence aggregation is the process of combining multiple messages into a single text plan. Whereas discourse planning structures the abstract messages, sentence aggregation performs the task of combining grouped messages into concrete text plans.

A simple caption-generation system can use a very simplistic form of aggregation: realizing each message as a separate sentence, for example, or combining all of the selected messages with a conjunction such as “and”.

Often the individual messages can be combined in various ways to produce more fluent text; for example, to describe the three increasing datasets in the example data, the final caption could have the form “the values for Quebec, British Columbia, and the Maritimes all increased”, or even simply “all of the values increased”. Other aggregations of this form can be done if all of the messages have common constituents of various types.

The domain or type of the data may also influence the possible aggregation techniques. Sophisticated aggregations making use of such information are possible: for example, if a system knows the provinces of Canada, then it could aggregate a list such as “Nova Scotia, New Brunswick, and Prince Edward Island” to simply “the Maritime provinces”. Forming sets in this way requires domain knowledge.

Aggregation should be driven not only by the nature of the data and the messages selected from it, but also by the needs of the user of the system. In the example above of multiple increases, the user might want each individual item mentioned (“Quebec, British Columbia, and the Maritimes”), or might prefer the more general statement in the second case, “all of the values”; the system should support this sort of choice.

Selecting which of the possible aggregations should be performed on a particular caption is a difficult task. One possible approach is to use a corpus of target texts to determine which types of aggregations occur most frequently, and then to create rules which produce those aggregations.

In many of the previous systems, sentence aggregation is combined with other steps in the generation process. For example, the process of creating GOSSIP's (Section 2.2.4) topic tree, described above under discourse planning, also performs the task of sentence aggregation. For example, if several users spent time running programs and editing files, then these users would be grouped together in the tree, with a single node storing the list of all the individual users. GOSSIP also performs some aggregations as the modified tree is traversed to produce the text.

SAGE's (Section 2.3.1) text generator makes the text less awkward by adding sentence transitions, making definite references to entities, and performing some forms of ellipsis. This combines sentence aggregation functions with the lexicalization and realization tasks described in the following sections.

3.3.4 Lexicalization and referring expression generation

The next stage in the generation process is lexical choice—choosing words to express the messages created by the previous processes. There are two types of lexical choice that a caption-generation system must make: choosing words to describe the messages extracted from the data (referring expression generation), and choosing words to express domain concepts (lexicalization). Reiter and Dale (1997) separate these two forms of lexical choice; however, the two tasks are very closely related in caption generation, and will be dealt with together here.

Lexicalization is the stage in the generation process in which linguistic style enters the picture. There are many different possible ways to express the change in Quebec's value in the example data: "Quebec healthcare spending increases", "the value of healthcare spending in Quebec rises", "Quebec is spending more on healthcare", and many others. If the value being measured is from a different domain, then yet other patterns may be available. At least a set of basic alternatives should be built into a caption-generation system so that it can generate its sentences; more complex patterns and varied words can also be added as needed in a particular situation.

While the data determines the general sort of assertion that should be made (e.g., increase, decrease, correlation), choosing among the alternative methods of expressing a particular trend should be driven by the needs of the user. For example, if he or she desires a more informal caption, then colloquial language such as “Quebec is spending more on healthcare” could be selected in preference to the more formal “Healthcare spending increases in Quebec”. Creating rules to choose among multiple possible techniques of expressing the same relation is best done through an analysis of target texts.

In addition to describing the trends detected in the data, a caption-generation system must choose words to identify a particular entity in the domain. In general text generation, this task also includes issues of referring to an entity when it comes up in a discourse multiple times—selecting appropriate pronouns or generating new “definite descriptions” when the context rules out the use of a pronoun. However, in the majority of captions, any entity will be mentioned at most once, so the crucial issue in caption generation is the selection of an appropriate lexical item to refer to it.

These two types of lexical choice are of course very closely related. The form that is chosen to describe the trends of the data constrains what sort of words can be used to describe the data—for example, if the verb “increase” is selected to describe an upward trend in the value for Ontario, then the description of that variable itself must be realized as a noun phrase such as “Ontario healthcare spending” or “healthcare spending in Ontario”. As well, the available domain vocabulary can constrain the ways in which the trends can be realized—for example, if the quantity being measured can only be expressed as a verb, then the trend cannot also be expressed as a verb.

Beyond the above syntactic relationship between the two processes, there can sometimes be an interaction between the lexicons used for the two tasks—this can occur when the entity which is being described requires particular words to describe it. An example of this is in the domain of weather forecasts, where a decrease in temperature would more likely be described by a verb such as “cool down”, rather than simply “decrease”.

The lexical choice techniques used by previous systems vary. The systems which are tied to specific domains also have available the domain vocabulary; the lexical choices they must make consist largely of selecting appropriate words and phrases to describe the messages extracted from the data. For example, Ana's (Section 2.2.1) text generator chooses and combines phrases from its lexicon which capture the meaning of the message to be expressed and satisfy rhetorical constraints, using domain-specific semantic, syntactic, and rhetorical knowledge.

STREAK (Section 2.2.5) must choose among different patterns for realizing the messages it discovers in basketball-game summaries; for example, the result of a game can be expressed in the main verb ("Chicago beat Phoenix") or as a prepositional phrase ("with a 99–84 triumph over Denver"). The pattern to use is determined by the floating facts which must be added and the revision rules which are used to add them.

The domain-independent approaches do not make a great deal of use of the domain knowledge, other than to fill slots in the output forms. TREND (Section 2.2.6), for example, uses templates and inserts appropriate verbs and adverbs into templates which are then sent to a linguistic realizer. SelTex (Section 2.3.2) follows a similar slot-filling method, although it uses an adaptation of Ana to generate its texts.

Of the existing systems, only Ana considers matters of style in selecting lexical items to express the trends detected. It allows the user to specify constraints on the syntax of the generated text (such as a desire for few subordinate participial clauses), which are then used when choosing phrases from the lexicon and combining them.

3.3.5 Linguistic realization

Once the lexical items have been chosen to express the concept of the message, the final step is to convert the conceptual representation into text; this is the job of the linguistic realizer. As in any other text-generation domain, the realization can be done by a special-purpose realizer built for the particular system, or an existing realizer can be used as a "black box" at the end of the generation process. Both approaches have merit.

If an existing realizer is chosen, then the result of the processing to this point must in be the input language of the realization system, and so the representations at all other levels of the system must provide all of the information that the realizer requires. If the input language of the selected realizer does not fit well with the architecture of the rest of the system, then time is wasted in conversion. If the realizer is particularly idisyncratic in its needs, then it could make the implementation of the rest of the system awkward.

A large advantage of a "home-grown" realizer is that there are no constraints on the representations that can be used at any of the other steps in the generation process; the realizer can be written to understand whatever structure is most convenient to the desired method of implementing the rest of the system. However, there is the equally large disadvantage that such a special-purpose system must be built essentially from scratch—a potentially time-consuming task and one which is tangential to the actual task of caption generation. Unless a great deal of time is invested in the creation of the realizer, it may not be able to handle all of the syntactic issues of the target language or languages, which may cause the captions to be inferior to those generated by a system using an external text realizer.

A system may combine linguistic realization with the two lexical-choice process described above if the format of the captions to be generated is simple enough that no external realizer is necessary; in this case, it could follow a procedure like the phrasal lexicon approach of Ana (Section 2.2.1), or the template-filling of TREND (Section 2.2.6) (although TREND does send its filled templates to an external realizer). A system may also perform all of the realization steps at once if the output of the discourse planning and aggregation stages is already in a format acceptable to the selected realizer.

Many of the previous systems have used existing text generators: STREAK (Section 2.2.5), TREND, and AutoBrief (Section 2.3.3) use FUF/SURGE (Elhadad and Robin, 1996) as a final step, while FOG (Section 2.2.3), LFS (Section 2.2.2), and GOSSIP (Section 2.2.4) all use a system which later developed into RealPro (Lavoie and Rambow, 1997), and SelTex (Section 2.3.2) uses FRANA, a French derivative of Ana.

3.4 Other necessities

During the discussion of generation tasks in the preceding section, the need arose for several other components of a caption-generation system: some knowledge of the domain from which the data is drawn, a model of the system user and of the audience, and integration with the system producing the graphics. The following sections describe each of these requirements in more detail.

3.4.1 Domain knowledge

A simplistic caption-generation system could be made to work using no more domain knowledge than that which is necessary to give the proper names to the variables in the input data. However, the captions created by such a system would be very generic, and potentially not useful in the particular domain and application for which it was being used.

Knowledge of the domain from which the input data was drawn should guide a caption-generation system through the entire process of producing its text, from the selection of "interesting" messages to include in the output to the final realization of its caption in natural language.

In different domains, different features of the data may be of interest; for example, in one case it may be of more interest to concentrate on the final value of a variable, while in another it could be the individual changes in its value from start to finish that is the most relevant feature.

The same data may even have different meanings depending on what domain it comes from, or even the perspective that is taken. In the example data presented in Figure 3.2, for example, the increase in spending in British Columbia could be viewed as good if the author's goal is to encourage more spending. In an article which advocated cutting government spending, on the other hand, the decrease in Alberta's spending might instead be highlighted.

The domain can also have an effect on how the messages are ordered once they have been selected. For example, in the FOG system (Section 2.2.3), messages are ordered strictly by salience when marine weather forecasts are being created; however, for public forecasts, messages are first grouped by temporal order and then by salience.

Once the messages have been extracted from the input data and organized as needed, the domain model should also be used in selecting an appropriate method of presenting the information textually. This procedure has two facets: appropriate words and phrases should be selected to describe the domain elements, and domain-specific ways of expressing the messages extracted from the data may also be necessary.

The first task, describing the domain elements, can be done very simply by providing a mapping between the variables in the input data and lexical items which can be used to describe them. However, it might also be necessary to select among alternative descriptions of the same element in different situations; in such cases, the domain model should provide rules for making those choices.

Selecting appropriate lexical items to present the messages extracted from the data could be done in a largely domain-independent way, using words such as “increase” or “decrease”. However, often there are words or phrases which are typically used in some domains to describe trends. For example, in the stock market domain for which Ana (Section 2.2.1) was written, phrases like “surrender a gain” and “posted a loss” are common, while in weather forecasts (FOG), winds “diminish to light”. Such differences should also be part of a system’s knowledge of its domains.

There are several possible ways that the necessary domain knowledge could be included in a caption-generation system. Probably the most straightforward method is to “hard-code” the necessary information into the system itself; this is the approach taken by the previous domain-specific systems, such as Ana, GOSSIP (Section 2.2.4), FOG, and STREAK (Section 2.2.5). This is the simplest way to ensure that the system has access not only to appropriate vocabulary, but also to rules for choosing messages and for selecting among different possible methods of realizing those messages. However, including the domain information in a system in this way makes it much more difficult to extend the system to work in new domains if necessary (although Robin and McKeown (1996) claim that a large number of the rules developed for the basketball-game domain ended up applying to other domains as well).

An alternative approach is to specify the domain information entirely in the input. This is certainly more flexible, and allows for the system to be used in any possible domain; this is the approach taken by SelTex (Section 2.3.2). However, while it is fairly straightforward to include the necessary lexical items in the input, encoding the necessary selection rules is much more complex and could well prove impossible for more complex domains.

A third possible approach, a hybrid which combines the above two, would be to provide a set of many possible rules for content selection within the system, and then to define the necessary domains in terms of those rules; the vocabulary could also be specified in the input as in the above approach. This allows for more complex rules to be specified on a per-domain basis and for the domain to be described on the fly instead of hard-coding it into the system. However, selecting an appropriate set of possible rules could prove quite complex, as could defining any new domains in terms of these rules.

A number of previous systems, such as Ana, FOG, and the project-management application of SAGE (Section 2.3.1), are tied to particular domains; naturally, they make use of a great deal of domain knowledge throughout the generation process. Ana even has as one of its basic tenets that domain-specific semantic, linguistic, and rhetoric knowledge is required for a computer to produce intelligent and fluent text. Such previous systems use their domain knowledge throughout the process.

The domain-independent systems still require some knowledge of the domain. The domain knowledge in SelTex, for example, consists of a lexicon and a characterization of the data's relational structure (i.e., the units of the variables and which can and cannot be used as relational keys); it is specified entirely in the input file. AutoBrief (Section 2.3.3) specifies its high-level goals in a domain-dependent manner and then decomposes these goals into domain-independent subgoals, using domain-specific knowledge to interpret the high-level goals.

3.4.2 User model

The final form and content of an automatically generated caption should not be a function simply of the data on which it is based and of the domain from which that data is drawn; the purpose for which the caption is being generated should also guide its generation.

When speaking of a "user model" in the context of captions, it is important to remember that there are two possible "users" of a caption-generation system: the graphic designer or journalist who requires a caption to include in a presentation, and the eventual target audience of that presentation. The needs of both of these users should be considered.

The user who is actually producing the caption will usually have some idea of what he or she wants the caption to express—for example, which data values to emphasize or which aspects of the data to concentrate on. He or she might know enough to be able to specify the precise techniques from the system's repertoire which should be used in producing the caption. In this case, the system may simply produce a presentation using the specified techniques and not need to consider the audience. However, the system user will not always be skilled at creating presentations; he or she might not always know the exact technique to choose in order to produce the right effect.

Instead of such specific techniques, the input from the system user could also consist of more abstract communicative goals for the caption ("intentions", in the language of Fasciano (1996)): for example, the user could want to persuade the audience that a particular company is doing well compared to its competitors. In some cases, the user of the system might not even know or care what the actual trends are in the data, but should still be able to specify what the system should look for in the data and what it should do with what it finds. These abstract goals must be mapped into specific presentation techniques.

In this case, it is crucial for the system to consider the audience of the presentation. They are the ones who will be viewing the final presentation, and the system should ensure that the result will have the right effect on them. The whole generation process should be geared towards having an appropriate effect on the audience; whatever goals the system user specifies, the generation process should meet those goals in creating the caption.

The system should have a library of possible presentation techniques which can be used in creating its captions—these techniques can be applied at all of the steps listed in the preceding sections, from the initial selection of content to the final textual realization step. They can specify which aspects of a dataset should be considered "interesting", how to combine information from different datasets in various contexts, and the word and sentence structures that should be used to express the final assertions; the techniques may be domain-dependent or domain-independent. A caption-generation system should have a variety of such techniques—as many as are required to produce the particular types of text that are required.

The possible different techniques which a caption-generation system might use should be derived from a corpus of texts of the sort the system aims to produce. The data and communicative goals used to produce the corpus texts should both be considered when deriving rules to use; any other available contextual information could also be useful in determining the reasons behind the choice of presentation technique.

The existing systems take a variety of approaches to selecting among the various presentation techniques. For example, PostGraphe and SelTex (Section 2.3.2) use a table which associates each possible user goal (such as *comparison*) with the schemata that can express it and a weight indicating the efficiency of each schema at expressing that intention. A planning algorithm then selects appropriate schemata to realize all of the user's goals. SelTex's various textual presentation techniques and the heuristics which choose one over another were gathered from an analysis of a variety of text-graphics pairs.

AutoBrief (Section 2.3.3) also uses a planning process to refine and decompose the user's domain-specific communicative goals into medium-specific actions. It is not clear how the rules used to perform this decomposition were derived; however, they aim to specify the techniques which are most commonly used to achieve particular goals.

Ana's (Section 2.2.1) fourth module performs this task; it selects phrases from the lexicon that capture the meaning of the messages and satisfy rhetorical constraints. The user may specify constraints on the system which determine the complexity and rhetorical structure of the generated text; the selection of phrases from the lexicon is guided by these constraints.

3.4.3 Integration with graphics

The eventual goal for a caption is to be presented alongside a graphic in order to point out important features of the data displayed on the graphic. So far, this chapter has described only the choices that must be made inside the text-generation component; however, a full caption-generation system should also consider the features of the graphic the captions are generated to accompany.

There are two aspects to this issue: the assertions to be made must be allocated among the available output media, and the various components of the resulting presentation must be well-integrated. Zelazny (1996), in a book aimed at business presenters, advocates first selecting the textual message which is to be presented, and then producing a graphic which supports that message. This is good advice for manually created captions, as people are good at choosing appropriate words to express the message they want to present; however, in an automated system, it is better to produce the various components of the presentation together, guided by a common goal.

Choosing how to allocate a particular message across the available media is a difficult task. Generally, it is known that graphics are superior for showing quantitative relationships and for presenting large amounts of statistical information, while text has the advantage of being able to stress particular aspects of the data; however, specific rules which can guide an automated system are not so easily stated.

The general question of allocating different components of a message among the available media has not been solved. According to Roth and Hefley (1993), most systems use domain-specific heuristics for making such selections, using criteria such as the number of relational facts to be presented or whether the information consists of physical attributes or abstract actions. The effectiveness with which each medium realizes the purpose of a presentations should also be considered.

Once the messages have been allocated among the available media, the task still remains to produce a coherent presentation from the individual pieces. If a graph-generation system and a caption-generation system use the same representation of the data, context, and user goals, it is unlikely that the final presentation will be completely unacceptable even if there is no interaction between the two systems. However, better results can be obtained if there is communication between the graphics component and the text component. For instance, if one of the goals is not possible to satisfy with graphics, then the graphics component could communicate to the text component that it should emphasize that particular aspect of the data. Also, the graphics generator could omit some information if it knows that it is already being mentioned in the caption.

As well, if there is communication between the two components, then the text can explicitly refer to physical features of the graphic (such as “the top line” or “the red bar”)—this enhances the integration across the media. The graphics generator could even indicate to the textual component that some part of the generated image may be difficult to understand, and a note to explain the graphic itself would be added to the caption. Making sure that the graphics and text generators produce compatible results could also be the task of an overall coordinating component, as in AutoBrief (Section 2.3.3). This component would translate the user’s goals into specific tasks for the two generators, making sure that the tasks are compatible.

Of the existing systems which generate integrated text-graphics presentations, only SAGE (Section 2.3.1) considers medium allocation; essentially, as much as possible is expressed in graphics, and the remaining ideas are generated in the text. In AutoBrief, the user specifies medium-independent goals which are translated into medium-dependent goals for the text and graphics generators. However, there is as yet no attempt made to distribute the messages to be generated between the medium-specific generators; each medium will express as much of the content as it is capable of expressing.

PostGraphe’s (Section 2.3.2) graphics generator and SelTex both use the same input data; however, once again, there is no effort made to allocate the messages across the media. If there is a graphical schema which will realize the intention, then it is used; if there is a textual schema which will also realize the intention, then it is used too. The textual schemata come from an analysis of a corpus of text-graphics pairs, so it is likely that they will mention the correct sort of information and not be overly redundant; however, there is no explicit allocation of components of the message to different output media.

3.5 Summary

A full caption-generation system should provide a framework in which a variety of rules can be implemented. Specifically, it should allow for rules to guide all of the generation tasks—content determination, discourse planning, sentence aggregation, lexicalization, referring expression generation, and lexical choice. These rules should be based on knowledge of presentation techniques, of the domain of the data, and of the goals of the user. Any particular system might implement only a skeletal version of any of these components; however, the system should be designed in such a way that it is straightforward to add new rules or new knowledge to produce different sorts of captions as the need arises.

A full caption-generation system should work with the component which is producing the graphics in order to create well-integrated graphics presentations; they may communicate directly, or some overall process may control the two generators. Various portions of the message should be allocated to the two media by some process which considers the capabilities of each medium to ensure that the result is coherent and does not contain any redundancies.

To perform these tasks, the caption-generation system should have knowledge of a variety of textual presentation techniques and rules to guide the choice among them in order to have the proper effect on the audience. It should also have some information about the domain from which the input data comes, to guide both its choice of information to present and the words and phrases it uses to present the selected information.

Chapter 4

CAPUT: A caption generation system

Chapter 3 outlined the theory underlying caption generation and the requirements for a system which generates such captions. Moving from the theoretical to the practical, this chapter describes CAPUT, a particular implementation of a caption generator.

CAPUT[1] is a system designed to generate single-sentence summaries of statistical data contained in an input file. The summaries generated are suitable for use as captions for graphics generated from the same input data. CAPUT was implemented in Java 1.1 (Arnold and Gosling, 1997), with CoGenTex's RealPro (Lavoie and Rambow, 1997) as the linguistic realizer.

4.1 Generation algorithm

The following is a sketch of the procedure CAPUT follows to generate a caption. Later, a particular Java implementation of this procedure is discussed.

The input to CAPUT is contained in a file that specifies a variety of information about the desired form and content of the caption. There may also be any number of datasets in the file, each of which can also have its own descriptive information. A simple tree is created to represent the content of the input file, with a root node and one leaf node for each individual dataset

[1] The name is not an acronym as such; the CAP refers to captions, of course, and UT could be thought of as a reference to the University of Toronto if some explanation must be given.

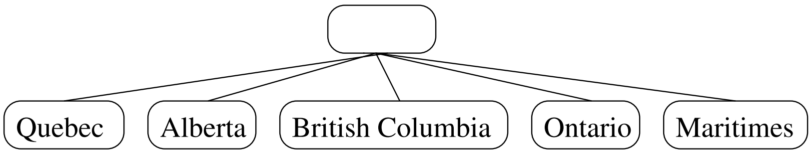


Figure 4.1: Initial tree structure—one node per dataset

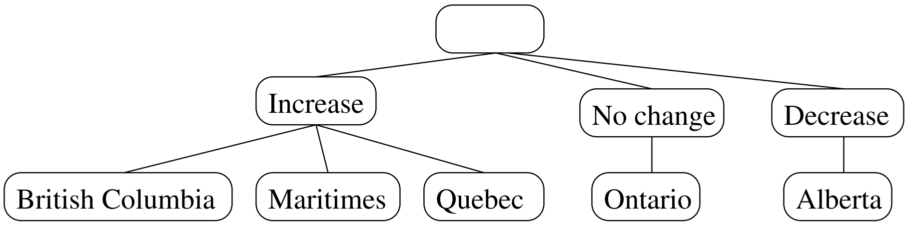


Figure 4.2: Revised tree structure—datasets sorted and aggregated

in the input. Descriptive information is attached at the appropriate level—to the root for common information, and to the leaves for information which differs across datasets. An example of an initial tree is shown in Figure 4.1.

This initial tree contains all of the information in the input file, in the order in which it was listed. However, quite often a successful caption will only mention some of the input data items, and it may sort or group the information which it does mention in a variety of ways. The next step in the generation process is to sort and group the datasets to produce a representation of the content of the final caption to be generated; this is accomplished by modifying the initial tree.

The modification of the tree may include the following steps: ordering the datasets, grouping the datasets, calculating totals or averages, or removing datasets that will not participate in the final caption. The criteria used to reshape the tree depend on both the nature of the data and the specification of the desired caption. This process advances in a top-down fashion, starting at the root of the tree and continuing until there are no more groupings to be made.

For example, if the data for British Columbia, the Maritimes and Quebec increased, that for Ontario remained constant, and Alberta's data decreased, and the selected grouping technique was to combine datasets with like parity, then the grouping would look like that shown in Figure 4.2 after this portion of the process is complete.

After the tree has been modified, the next step is to generate a text plan for the caption. The plan is created from the tree in a bottom-up fashion; each subtree generates a plan fragment which represents its content, and these fragments are then combined at the parents until the root node has the complete text plan.

The final step is to send the text plan to the linguistic realizer. Currently, there is no automatic generation of the graphic which the caption is to accompany, so the output of CAPUT is simply the English version of the caption.

4.2 Implementation

The procedure described above was implemented in a Java prototype. This section describes various features of the current Java implementation of CAPUT: the important classes, and the other components such as the format of the input file and the lexicon and the text realizer used.

4.2.1 Important classes

CAPUT was implemented in Java 1.1 (Arnold and Gosling, 1997). The tasks of modifying the tree and producing the text plan from the result are performed by a number of different Java classes; for example, a particular class extracts the information from the datasets, while another groups the datasets in the tree, and still another combines the information from the tree into a text plan. In a particular caption-generation setting, specific subclasses which perform their respective tasks in a particular way are used to ensure that the content and form of the caption are appropriate to the situation. The following sections provide details of the main classes used; Appendix A provides a hierarchy of all of the classes in the current implementation.

TreeNode

Central to CAPUT's caption-generation process is its tree of information. It is created from the initial input file, and all of the subsequent processes act on this tree to produce the final caption.

All of the nodes of this tree are instances of the TreeNode class.

There are two distinct sorts of TreeNodes: LeafNodes and ParentNodes. As their names suggest, these two types of nodes play two different roles in the tree; LeafNodes are at the edge, while ParentNodes form the interior of the tree. A LeafNode contains a single dataset and the context information associated specifically with that dataset.

ParentNodes also contain context information; any information attached to a ParentNode implicitly applies to all children of that parent. As well, ParentNodes have a number of associated objects that help to reshape the subtree rooted at the node and to produce a text plan representing that subtree. Every ParentNode must have an Aggregator, and it may have a MessageExtractor and a Template as well; if a particular ParentNode does not have one of these, then it uses the object attached to its parent. This means that the root node of the tree *must* have all of these objects. These component classes are described below.

Field

Field is the abstract superclass of any sort of real-world context information which can be attached to a node in the tree. Information attached to a ParentNode is inherited by its children and can be overridden by information explicitly attached to the child. Each Field subclass fills a specific slot on the node—for example, the Action class fills the *action* slot. Currently, the subclasses of Field are Subject, Action, and DirectObject.[2]

If more than one Field wants to fill the same slot on a node, then all of the Fields are stored temporarily on that node. Then, when the information from the node is actually needed, all of the other information about the nature of the desired caption is used to help choose the appropriate Field to use. For example, if the input specifies that the action should be realized as a noun, but one of the possible Actions specifies only a verb form, then that Action can be eliminated from consideration. This resolution is performed by the particular instance of Field itself

[2]Confusingly, an instance of DirectObject can also represent an indirect object; this name was chosen to avoid clashing with the built-in java.lang.Object class.

Field	Parameter	Value
[consume]	verb	CONSUME
	noun	CONSUMPTION
	direct	DIRECT
[Canada]	noun	CANADA
	adjective	CANADIAN1
[watermelon]	noun	WATERMELON

Figure 4.3: Three example Fields

whenever it is necessary, so that (for example) Subjects know how to resolve conflicts for the *subject* slot. If necessary, subclasses of the particular types of Field can also be created in order to make more sophisticated choices.

The Field instances contain the information needed to lexicalize the concept they represent in a variety of surface forms—the specification of the [consume] Action, the [Canada] Subject, and the [watermelon] DirectObject are shown in Figure 4.3. The *direct* field on [consume] indicates that it takes a direct object. Note that the parameter values such as CONSUME are RealPro lexicon entries, not actual words.

Dataset

A Dataset object contains the points constituting one dataset. It also contains a number of methods which serve to extract various information from the dataset, for use in grouping the datasets and generating the text plan. For example, there are methods to return the highest value in the set or the difference between its initial and final values. Additional methods can be added as required by other system components.

ActionSpec

An ActionSpec represents a single action that is the subject of one of the datasets—for example, the consumption of watermelons in Canada. In its simplest form, an ActionSpec has a subject, a verb, and an (optional) object, which may be direct or indirect; each of these components is an instance of a Field subclass. ActionSpec is a convenient way of encapsulating the context

Action:	[eat]
Subject:	[Canada]
Object:	[watermelon]
Complements:	[between [1990, 1998]]

Figure 4.4: Sample ActionSpec

information associated with a Dataset for use further on. An ActionSpec is created from each LeafNode during the time that the text plan is being created. Figure 4.4 shows the ActionSpec for the above example of watermelons in Canada.

An ActionSpec can also have an unlimited number of prepositional complements; these may be specified in the input file along with the other descriptive information, or may be added based on the aspect of the data on which the caption is concentrating. For example, if the caption is considering the time period from 1990 to 1998, a complement representing "between 1990 and 1998" would be added to the ActionSpec, as in Figure 4.4.

An ActionSpec can be realized in a variety of ways; the example above, the consumption of watermelons in Canada between 1990 and 1998, might become the text "Canadians eat watermelons between 1990 and 1998" or the noun phrase "Canadian consumption of watermelons between 1990 and 1998". The form that is chosen is determined by the selected Template at the time that the text plan is being generated. The Template will add to this text plan any relevant information about the value of the action being measured, such as whether it increased or decreased, to produce the final text plan; see the description of Template on page 62 for more information.

The information associated with several Datasets can be combined into one ActionSpec if they are compatible—that is, if they differ in exactly one of their components. For example, a single ActionSpec could represent "American and Canadian consumption of watermelons" or "Canadian consumption of watermelons and grapefruit". See Section 5.3 for examples of this process.

Aggregator

Aggregator is an abstract class which is never directly instantiated; it is the subclasses of Aggregator that do the actual work. An instance of an Aggregator subclass is attached to every ParentNode in the tree. An Aggregator has two main functions: to sort and group the children of the ParentNode with which it is associated, and to create a single text plan based on the information in the children of that node and corresponding to the subtree rooted at its associated ParentNode.

Different subclasses of Aggregator can perform very different transformations on the tree. For example, the ParityAggregator groups the children by the "parity" of the change in their values (whether the values increased, decreased, or remained the same), as in Figures 4.1 and 4.2. The ComparisonAggregator sorts the nodes and then uses heuristics to determine which elements to mention (for example, the highest or lowest value). The TotalAggregator and AverageAggregator compute total or average values, respectively, of the variables represented in the datasets. The Aggregator stores information about the choices it makes at this stage so that the following stage can produce the appropriate text.

When producing the text plan from the subtree with which it is associated, an Aggregator will extract information from each of its child nodes and combine them in whichever way is appropriate. The information from the child nodes could consist of fragments of text plans which are then combined using simple conjunctions such as "and" or "but"; the Aggregator might also use the ActionSpecs or the data from its children to produce a text plan from scratch. Which of these alternatives is used depends on the type of Aggregator and possibly on the nature of the data. It may use the information stored in the previous step to guide this choice; for example, if one child node remains, it must know whether it had the highest or lowest value, or was the only input to begin with, or had some other property which caused it to remain in the tree after the first pass.

The type of Aggregator to use on the root node is specified in the input file. Any new internal nodes created during the process of reshaping the tree are given an Aggregator of a type determined by the original Aggregator.

MessageExtractor

Like Aggregator, MessageExtractor is an abstract class whose subclasses do the work. A MessageExtractor extracts information from Datasets; this information is then used by the Aggregator to group and sort the Datasets. Different subclasses of MessageExtractor extract different information; for example, an IncreaseDecreaseKey extracts the difference between the starting and finishing values of a dataset, while a SingleValueKey uses a single point from the dataset. See Chapter 5 for examples of various MessageExtractors being used.

A MessageExtractor also performs the task of choosing an appropriate Trend object to be used in the creation of the text plans. Each type of MessageExtractor has its own procedure for selecting a Trend; it uses the information from the datasets, and may also query the Aggregator for information about the choices it made to guide this process. For example, an IncreaseDecreaseKey chooses an IncreaseDecreaseTrend if there was only one child node to start with; this would result in sentences stating that something "increased" or "decreased". It chooses an IncreaseDecreaseCompareTrend if there were multiple children which are being compared, which results in sentences containing phrases like "increased the most". A subclass of MessageExtractor can perform arbitrarily complicated operations to choose an appropriate Trend.

The final task of a MessageExtractor is to attach the appropriate prepositional complements to the ActionSpecs, so that this information can be included if necessary in the final caption. The IncreaseDecreaseKey adds the starting and ending dates of the data in the dataset, for instance, while the PercentageChangeKey also inserts the percentage increase or decrease.

Trend

A Trend represents the trend that was detected in a particular Dataset by the MessageExtractor. Once again, Trend is an abstract class; it is the subclasses of Trend that implement the actual process of describing the trend. For example, a SingleValueTrend describes the fact that a single point or a set of single points had the highest or lowest values, resulting in generated text of the form "Canadian watermelon consumption had the highest value".

Trends can generate text plans in the form of a verb phrase (e.g., "increased the most"), an adverb (e.g., "less often"), or an adjective (e.g., "more"). Specific types of Trends might not be able to generate all of these forms; if it cannot produce the requested form, the Trend will throw an exception to indicate that the type of caption specified in the input cannot be realized. The selected Template determines which sort of text plan the Trend should create and integrates the result with the text plan from the ActionSpec.

The type of Trend to use is not specified in the input; rather, it is selected by the MessageExtractor based on the data and possibly the state of the Aggregator, as specified in the description of the MessageExtractor class.

Template

A Template converts a set of ActionSpecs and a Trend into a text plan. Different subclasses of Template will put the information together in different ways. For example, if the action that was measured was Canadian watermelon consumption and it increased, then a NounTemplate would realize it as "Canadian consumption of watermelon increased", with the action as a noun phrase and the trend as a verb. By contrast, a VerbTemplate would produce "Canadians eat more watermelons" on the same input; in this case, the action is a verb phrase and the trend is the adjective "more".

The input file specifies the type of Template to use in generating the caption. Syntactic information about the final caption form can also be specified as arguments to the Template—for example, the input could request a caption in the past tense or the progressive mood. Examples of the use of different Templates with various arguments are presented in Chapter 5.

```
template: NounTemplate
extractor: SingleValueKey
aggregator: ComparisonAggregator
context: buy DirectObject( WATERMELON )
-----
context: australia
1998    1250

context: canada
1998    1500
```

“Canadian consumption of watermelon in 1998 has the highest value.”

Figure 4.5: A sample CAPUT input file and its output

4.2.2 Other components

In addition to the classes described above and their subclasses, CAPUT includes a number of other components. The input to the system comes from an input file in a particular format; the system has access to a lexicon; and it uses an external realizer to produce the final form of the text. This section describes each of these components of CAPUT.

Input file

The input file specifies the data from which the caption is to be generated, as well as a variety of information to guide the choices made during the generation process. A sample input file and its corresponding output are shown in Figure 4.5.

There are two parts to an input file: the preamble and the individual datasets. The preamble, which gives the specification for the type of caption which should be generated, has four fields. If any of these fields is omitted, then a generic default is used. The *template* field determines which template should be used for the caption—for example, whether the action which is being measured should be realized as a noun or as a verb phrase.

The *extractor* field specifies which aspect of the data to use for grouping or sorting. In the example, a SingleValueKey is specified; this uses the value at a single point, rather than some other aspect of the data such as the overall increase or decrease.

The keyword *aggregator* declares which Aggregator should be used to group the datasets and to combine their assertions. A ComparisonAggregator, specified in this example, uses heuristics to choose the highest or lowest values, depending on the characteristics of the data.

Finally, the *contexts* field provides a list of contexts to guide lexical choice. These contexts can be named concepts from the lexicon, such as *buy* in the example—see the following section for a discussion of the lexicon. They may also be the specifications of individual Fields, such as the *watermelon* DirectObject. All of the contexts listed at this point in the input file are attached to the root ParentNode of the initial tree.

The second section of the input file consists of the individual datasets. Each dataset has its own list of contexts, which can once again be either named concepts from the lexicon or the specifications of Fields. These contexts are attached to the LeafNode corresponding to a particular dataset. Each dataset also specifies a list of (x, y) pairs which describe the data.

All of the classes (such as Template or Aggregator) that can be directly specified in an input file are instantiated by the same process. First, an instance of the desired class is created, and then any specified arguments are passed to that instance. So that this can be done, all such classes inherit from BasicObject, a class which provides methods for setting the parameters once the class has been created; the classes must also provide a no-argument constructor so that they can be dynamically instantiated.

Lexicon

CAPUT's knowledge of the lexicon of the domains about which it is generating captions comes from a list of "contexts" which it reads in on startup. These contexts specify appropriate lexical items to use in a particular context—for example, the context in Figure 4.6 shows the appropriate words to use when the topic is consumer purchases or Canada. Any words in all-capitals specify entries that appear in RealPro's lexicon (see the following section for more on RealPro), while words in lower case are unknown to RealPro. If a word is not in RealPro's lexicon, then its syntactic category must be specified in the text plan when it is used so that the caption can be realized.

```
// purchasing context
buy {
    Action( CONSUME, CONSUMPTION, DIRECT )
    Action( BUY, DIRECT )
    Action( PURCHASE1, DIRECT )
}

// Canadian context
canada {
    Subject( CANADA, CANADIAN1 )
}
```

Figure 4.6: Sample lexical knowledge base concepts: *purchasing* and *canada*

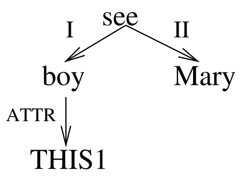
When the file is read, the lexical items are instantiated using the same method as the classes specified in the input file (see the previous section). As well, each lexicon context (such as *canada*) is stored in a hash table under its name so that the lexical items in that context can be retrieved if the context is specified in the input file.

Each context in the list specifies one or more lexical items. When one of these contexts is specified in an input file, the associated lexical items are attached to the appropriate TreeNode. If there is more than one item attempting to fill the same slot during the generation process—such as with the three Actions listed under "buy"—then the conflict is resolved as specified above in the description of the Field class.

Text realizer

CAPUT uses CoGenTex's RealPro text realizer (Lavoie and Rambow, 1997) for the final step of linguistic realization. RealPro is a fast, portable linguistic realizer implemented in C. It developed out of the text realizers used by systems such as LFS, FOG, and GOSSIP (see Chapter 2).

The input to RealPro is the Deep-Syntactic Structure—"DSyntS" for short. This structure is based on Mel'čuk's Meaning-Text Theory (Mel'čuk, 1988). A DSyntS is an ordered tree with labelled nodes and arcs. Every node is labelled with an uninflected lexeme from the target language (in this case, English); there are no non-terminal nodes. The arcs are labelled with syntactic relations such as *subject* rather than semantic relations like *agent*. No function words



"This boy sees Mary."

Figure 4.7: An example of DSyntS

```
SEE [ ] (
  I  BOY [ article:def ] (
    ATTR THIS1 [ ]
  )
  II Mary [ class:proper_noun ]
)
```

Figure 4.8: ASCII representation of sample DSyntS

are represented in the tree (thus "deep" syntactic structure); it consists only of meaning-bearing lexemes. The input to RealPro fully determines the output, but it specifies it at an abstract level. An example of a DSyntS (adapted from (Lavoie and Rambow, 1997)) is shown in Figure 4.7. Note that if the feature **question:+** is added to the verb and **number:pl** to the **boy** node, then the resulting text is "Do these boys see Mary?" This illustrates that function words do not need to be included in the input DSyntS and that syntactic issues are handled automatically. Figure 4.8 shows the ASCII representation of the DSyntS of Figure 4.7; for the remainder of this thesis, the ASCII representation will be used whenever DSyntS is displayed.

The tree structure of DSyntS is based on the notion of syntactic dependency. Broadly speaking, each lexeme in the sentence "depends" on exactly one other lexeme; in other words, the dependent lexeme is present in the sentence because of the presence of the lexeme it depends on. There is exactly one lexeme which is not dependent on any other; in a full sentence, this will always be the main verb. This independent lexeme forms the root of the tree.

As the sentence is generated, the DSyntS is transformed into a Surface-Syntactic Structure (SSyntS); here, the abstract dependency relations used in the input such as *subject* are transformed into concrete relations such as *predicative*. The next step is to transform the SSyntS into a Deep-Morphological Structure (DMorphS). This is done by using rules of ordering of governors and their dependents and of dependents at the same level, and by adding default features to the lexemes. Then a Surface-Morphological Structure (SMorphS) is created by converting the abstract lexemes into their surface representations. Next, a graphical component adds abstract punctuation and formatting instructions to the SMorphS to produce the Deep-Graphic Structure (DGraphS). Finally, formatters transform the DGraphS into formatting instructions for the targeted output medium, which is currently one of ASCII, HTML, or RTF.

RealPro can be run in the background as a generation server; DSyntS can then be sent to it, and it will generate the text on the fly. RealPro comes with a Java interface, which allows DSyntS to be created and sent to the server programmatically. This is how RealPro is used in CAPUT.

The main class in the Java API to RealPro is CGT_SyntNode, which represents a deep-syntactic node with or without dependents. A DSyntS is represented in Java by a number of CGT_SyntNodes arranged in a tree structure; it can be manipulated as needed during the generation process to produce an appropriate plan for the final text. Once the final DSyntS has been created, the root CGT_SyntNode is sent to the server for realization via the CGT_Realizer-SocketClient class, which connects to a running RealPro server, sends the DSyntS, and returns the string of generated English text.

4.3 CAPUT and generation principles

Chapter 3 described a set of principles which a caption-generation system should follow. This section examines CAPUT in the context of those principles. While CAPUT does not implement all of the principles outlined in that chapter, it does consider a number of them and can be extended to be more comprehensive in the future.

4.3.1 Generation tasks

Section 3.3 presented Reiter and Dale's (1997) list of generation tasks, and proposed how each task should be addressed by a system to generate captions for information graphics. This section considers CAPUT's approach to each of these tasks.

Content determination

Content determination consists of selecting the relevant information from each dataset in the input. In CAPUT, this task is performed by the specific type of MessageExtractor specified by the system user in the input file. The various types of MessageExtractors currently implemented in CAPUT choose different information from the datasets—for example, the absolute increase or decrease in value, the percentage change from start to finish, or the value of a variable at a specific point.

In the present implementation, CAPUT does not have any knowledge about the effect of selecting different features from the data or any rules to choose among the possibilities. It is the user who must specify the MessageExtractor in the input, so it is the user alone who considers issues such as the graphic the caption is to accompany, the domain of the data, and the desired effect on the audience. In future versions of the system, these issues can be considered—see Section 6.3.2 for proposed implementations.

The set of MessageExtractors in CAPUT can easily be extended to allow the generation process to concentrate on new aspects of the data. A new MessageExtractor requires that a programmer specify the following: a "key" which can be used by the Aggregators to sort and group the data; one or more new types of Trend which will produce appropriate output from the data selected by the MessageExtractor; and a procedure for creating an appropriate Trend object given a dataset and the state of the Aggregator.

This basic method of determining content—extracting information of a particular type from each dataset—is, of course, not novel. However, the fact that it is done by special-purpose classes instead of by a built-in part of the system means that it is more flexible than other methods which build the selection rules into the core of the system itself. This method is domain-independent in general, and domain-specific classes can be created to implement rules such as the 120 used by Ana (Section 2.2.1) in the stock market domain.

Discourse planning

Discourse planning is the process of structuring the messages produced by the content determination step into a coherent text. In CAPUT, this task is performed by the Aggregators during the process of converting the initial tree into a representation of the content of the caption.

The type of discourse planning performed on a particular caption is determined by the type of the Aggregator on each parent node. The top-level Aggregator is specified by the user in the input file, while the type of any sub-Aggregators is determined by the type of the original. As in content determination, therefore, it is currently the task of the user to decide what form of planning is to be done; there is no consideration of the other factors. Once again, Section 6.3.2 presents possible enhancements to CAPUT to allow the system to perform these tasks.

The various Aggregators currently implemented in CAPUT perform a wide variety of aggregations on the descendants of the node with which they are associated. The simplest Aggregator, a BasicAggregator, does not modify the tree at all, while a SortAggregator simply sorts the children by the value of their sort key. A ParityAggregator groups the children by the parity (positive, negative, or zero) of its sort key, while a TotalAggregator computes totals grouped by a specified field.

This framework allows for aggregation rules as complex as required to be included. For example, Corio's (1999) rules about which element(s) to mention in a comparison (e.g., the highest, highest two, or lowest) are implemented in the ComparisonAggregator. To add new rules would require the creation of a new Aggregator subclass, and possibly the addition of new methods to other classes, especially Dataset and MessageExtractor, so that the new Aggregator could access the sort of information it needs in order to structure the children of the node it is attached to.

It is possible that certain Aggregators do not interact well with particular MessageExtractors. For example, it makes no sense to use a TotalAggregator with a PercentageChangeKey; this would produce a caption that talked about the total percentage increase or decrease across datasets, a meaningless value. Aggregators or MessageExtractors can verify that they are compatible before attempting to produce a caption, and possibly output a message to the user or fall back to a generic implementation if the incompatibility is profound.

This method of discourse planning is inspired by that used in GOSSIP (Section 2.2.4) and LFS (Section 2.2.2). The nodes in GOSSIP's topic trees contain more and more varied information than those in CAPUT's trees. For example, the nodes may have labels such as *res-cons* (resource consumption) or *int-period* (interactive period during the session), while the arcs connecting the nodes represent conceptual links such as aspect, agent, or action. In the case of CAPUT, the data to be represented is less structured than the audit trails used by GOSSIP, so the simple tree of datasets and dataset groupings is sufficient.

Sentence aggregation

Sentence aggregation translates a set of conceptually-grouped messages which are the output of the discourse planner into a single overall text plan. This task is executed on the second pass through the modified tree, when the actual text plan is being created.

The Aggregator on each parent node performs the task of combining the information in each of its child nodes into a text plan. How this is accomplished is determined by the type of the Aggregator. For example, a BasicAggregator or SortAggregator will simply combine the DSyntS produced by each of its children with the conjunction "and", while a ParityAggregator will perform more elaborate transformations.

The more sophisticated Aggregators may attempt to combine the ActionSpecs from each of their children into a single ActionSpec; currently, this is possible only if the ActionSpecs differ in exactly one feature. An example of this process is given in Section 5.6.

Sentence aggregation follows a similar process in all of the preceding systems which address this issue; concepts are grouped together if they are sufficiently similar. When the data itself is more complex, then the meaning of *similar* changes, as in GOSSIP (Section 2.2.4); however, the idea remains the same, and CAPUT will still be able to perform aggregation given suitable classes to perform the comparisons.

Lexicalization and referring expression generation

Lexical choice, choosing words to describe the messages extracted from the data, is performed on the second pass through the tree, after it has been reshaped during the first pass. There are two sorts of choices that must be made in this step: choosing words to describe both the trends themselves (lexicalization) and the domain elements which make up the trends (referring expression generation).

The description of the trends is performed by several different components in CAPUT. The Trend selects the appropriate manner to express the features of the data; the type of Trend to use is selected by the chosen MessageExtractor, using the features of the data and the state of the Aggregator. For example, if the MessageExtractor measures a single value and it appears in the caption because its value was among the highest, then the eventual text produced would be of the form "... had the highest value". The Template specifies the overall syntactic form of the caption and integrates the results of the other processes into a single text plan. The type of Template to use is specified by the user in the input file.

In the current system, each Trend has at most one way to realize its information in the context of a given Template. In other words, if the message to be expressed is that the value of a variable had the largest increase and the Template specifies that the Trend should be realized as a verb, then the verb phrase "increase the most" will always be generated. In the future, the user should be able to specify features which would allow the system to choose among a larger variety of ways to express a message.

In addition to the trends detected in the data, the data itself must be described appropriately in the caption. As is the case when describing the trends, the Template specifies the desired part of speech for each component; the individual subclasses of Field then provide the required lexical items. This process is mediated by the ActionSpecs, which gather all of the context information (Field subclasses) associated with each grouping of datasets.

Some selection is done among the possible lexical items which could be used to describe an entity; if there is more than one possible word to describe it, then the conflict is resolved by the Field subclasses themselves, using the rest of the information about the caption to guide the choice. This process could be extended in future versions to provide more alternatives and more sophisticated methods of distinguishing between them.

The possible domain vocabulary can be specified in CAPUT's lexicon, or the lexical items may be directly indicated in the input file. New items can be added very simply to the input file, or if they are to be used multiple times, to the lexical knowledge base. Words with irregular forms (such as the verb "to eat") must also be added to RealPro's lexicon if they are not already there, so that sentences using them will be syntactically correct. The capabilities of the lexical items could be extended if needed in order to enable more complex selection processes among them.

Both of these forms of lexical choice take place at the same time in CAPUT, each time that a text plan fragment must be created for a message. The Template specifies the slots that must be filled, and the ActionSpecs and Trends together provide the text plan fragments to fill those slots. Additional syntactic information can be specified in the input file to further guide the generation process—for example, the required tense of the verb in the final caption.

The approach to describing the trends is more flexible than that used by some of the previous systems; for example, TREND (Section 2.2.6) uses very inflexible templates to produce its texts. Others are more flexible; STREAK (Section 2.2.5), for example, chooses among different parts of speech to express its facts depending on how they are to be inserted into the output, in a process similar to that employed by CAPUT.

When it comes to describing domain entities, only STREAK among the previous systems chooses from among a set of alternatives; its choice depends mostly on the syntactic role that the entity is to play in the output. Other systems either have the domain vocabulary built in (as in FOG (Section 2.2.3) or Ana (Section 2.2.1), for example) or else specify it in the input to the system (SelTex, Section 2.3.2).

Linguistic realization

CAPUT uses RealPro for the final realization step, which means that many intermediate results are stored as DSyntS. However, it is not tied to RealPro; any other linguistic realizer could be used as well. A Java interface to the realizer would be required in order to integrate it with the rest of CAPUT; all of the methods in CAPUT which create DSyntS would then have to be rewritten to use this new representation. However, the process of creating and reshaping the tree is not at all dependent on the form of the output and would not be affected by a change in linguistic realizer.

RealPro developed from the text realization systems used in LFS, FOG, and GOSSIP (Sections 2.2.2–2.2.4); these systems use a representation similarly based on Meaning-Text theory for the input to the realizer. Other systems also use existing realizers, including SAGE (Section 2.3.1), AutoBrief (Section 2.3.3), and STREAK (Section 2.2.5), all of which employ FUF/SURGE. Some of these (e.g., STREAK) use the input language of the selected realizer throughout the generation process; others, such as AutoBrief, use an internal conceptual representation which is then translated into the appropriate form for the actual realization step. CAPUT follows this latter paradigm, as it was more straightforward to implement the process of modifying the tree on custom-built structures.

4.3.2 Other necessities

In addition to the generation tasks described above, a caption-generation system also has other requirements: knowledge of the domain of discourse, a model of the audience, and the ability to be integrated with a graphics generator. This section describes CAPUT's approach to these requirements.

Domain knowledge

The domain knowledge of CAPUT currently consists only of a domain-specific lexicon, which specifies words appropriate for use in a variety of predefined domains. For example, when the data has to do with the consumption of food, words such as “consumption” and “eat” are appropriate.

The current implementation of domains could be extended to contain more information; for example, domains could specify appropriate Aggregators or MessageExtractors to use, or particular language which can be used to describe the trends. A weather domain, for instance, could indicate to the system that words like “hot” and “cold” are to be preferred when talking about extremes, rather than the generic “high” and “low”.

This use of a domain only as a lexicon is similar to the approach taken by SelTex (Section 2.3.2), although it is somewhat more flexible. Ideally, the domain model should be extended until CAPUT is capable of domain-specific reasoning similar to that employed by the domain-dependent systems such as Ana (Section 2.2.1) or SAGE (Section 2.3.1).

User model

The only user that CAPUT currently deals with explicitly is the actual user of the program—the person who is producing the caption. This user can specify a variety of constraints on the style and content of the caption, within the limits of what is implemented in CAPUT. CAPUT relies on this user or on some other external source to guide the selection of Aggregator, Template, and MessageExtractor; if none is specified, then it uses a generic default.

The components of CAPUT do have some knowledge of their own. For example, the ComparisonAggregator uses heuristics developed in Corio (1999) to select the appropriate element or elements to concentrate on when a caption comparing several values is to be produced. However, this knowledge is implicit in the rules themselves, rather than explicitly represented in a form which CAPUT reasons about. Other than the knowledge implicit in the implementation of some of its components, CAPUT does not have any information about the effect of any of its presentations on the audience.

The goal of CAPUT is to generate textual summaries of data that are suitable for presentation alongside a graphic produced from that same input data. Given this, it is possible that some overall coordinating system will process the top-level goals and transform them into specific goals which can then be expressed as input parameters to CAPUT, as well as specific goals for the graphics system; this is the technique used by AutoBrief (Section 2.3.3). If so, then CAPUT would not have to consider issues of the user model. The planning models used in SelTex (Section 2.3.2) and AutoBrief are probably the best method of integrating presentation knowledge into CAPUT; this model lends itself well to planning the graphical component of the presentation as well. Integration with graphics is discussed more fully in the following section.

Integration with graphics

CAPUT was designed to produce just the text portion of an integrated multimodal presentation. It does not explicitly consider issues of medium allocation and coordination across media, although (as mentioned above) it has some implicit knowledge in its rules for content selection (which were derived partly from actual text-graphics pairs).

A skilled user could use CAPUT as a tool in the process of preparing presentations, possibly alongside a system which generates graphics automatically, or with the graphics being produced by hand. If the graphics and the data use the same underlying data, it is likely that the caption will not be bad.

However, better results could be obtained if CAPUT were integrated with some graphics generator. The model could be that of AutoBrief, with a top-level controlling program driving the two individual generators; the two generators could also run in parallel and communicate with each other, as in SAGE (Section 2.3.1). Section 6.3.2 discusses integration with a graphics generator in more detail.

4.4 Summary

This chapter has presented CAPUT, a particular implementation of a caption-generation system. CAPUT has several novel features. It creates an initial tree representing the input data, and then chooses the content of its captions by modifying this tree. CAPUT is made up of a number of abstract classes, the subclasses of which produce a variety of behaviour. New subclasses can easily be added if new types of captions are required.

CAPUT addresses all of the principles described in Chapter 3 to some degree. Although it does not currently incorporate any presentation knowledge of its own, such knowledge is easily added. CAPUT can also be integrated with a graphics generation system to produce coordinated multimedia presentations; all that is needed is an intermediate layer to convert the end-user's rhetorical goals into settings of CAPUT's input parameters.

Chapter 5

Examples

This chapter presents several examples which demonstrate the capabilities of CAPUT. The first few examples produce simple enumerations of the features of the input data, while the later examples perform more involved manipulations to produce text that is more like a caption than a summary.

5.1 Example 1: Basic

Example 1 shows a very simple caption. The input file[1] for this example is shown in Figure 5.1, and Figure 5.2 shows a graph of the data.

The first step in the caption-generation process is to read in the input and create the initial tree to represent it. The tree created for this example is shown in Figure 5.3.

Once the tree is created from the input, the next step is to use the chosen Aggregator and MessageExtractor to reshape it. However, in this case, the user has specified a BasicAggregator, which does not modify the tree. So, in this case, the first pass through the tree does not change anything.

The next step is to generate the fragments of DSyntS for each of the subtrees, using the

[1] The data for all examples is fictional.

```
template: VerbTemplate
extractor: IncreaseDecreaseKey
aggregator: BasicAggregator
context: spend DirectObject( healthcare )
-----
context: quebec
1990    1400
1998    1450

context: alberta
1990    1500
1998    1400

context: british_columbia
1990    1610
1998    1900

context: ontario
1990    1700
1998    1700

context: maritimes
1990    1575
1998    1640
```

Figure 5.1: Sample input file

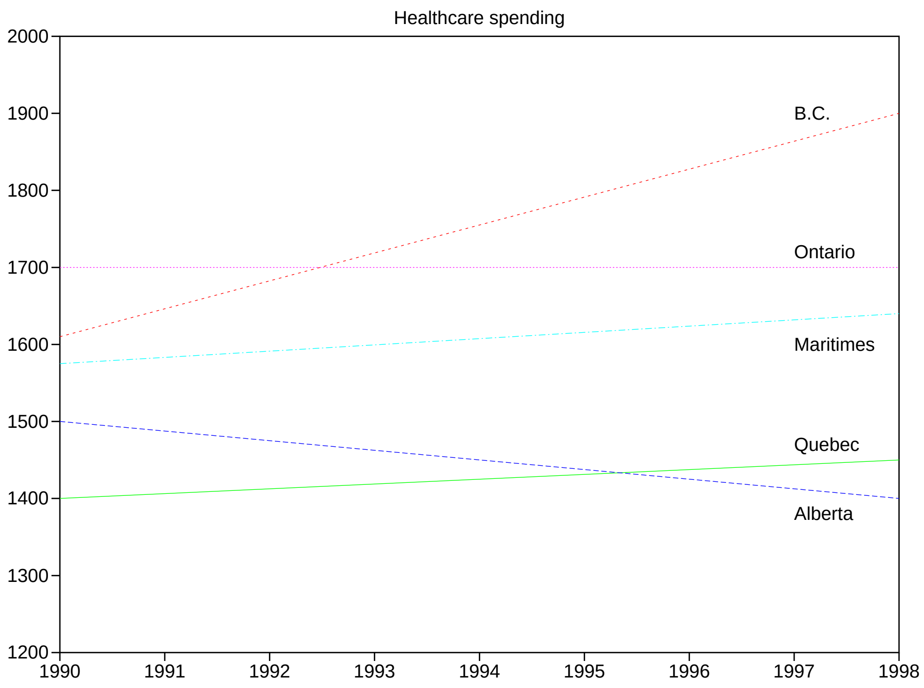


Figure 5.2: Graph of the sample data

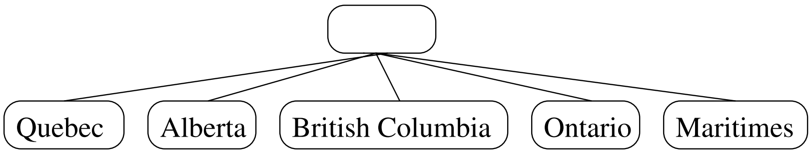


Figure 5.3: Initial tree for Example 1

Action:	[spend]
Subject:	[Quebec]
Object:	[healthcare]
Complements:	[between [1990, 1998]]

Figure 5.4: Action specification of "Quebec" node

MessageExtractor and the Template. We will concentrate the "Quebec" node for this example, but the same process takes place on the other nodes of the tree as well.

First, an ActionSpec is created from the information in the dataset on the "Quebec" node. This ActionSpec is shown in Figure 5.4. At this point, the entries in this ActionSpec are not yet lexicalized; they represent only concepts.

An appropriate Trend object is also created for use in the generation of the DSyntS; the type of this Trend is selected by the MessageExtractor and is determined by the nature of the data. In this case, the MessageExtractor is an IncreaseDecreaseKey and the data for Quebec shows an increase, so an IncreaseDecreaseTrend with a positive direction is created.

Once the ActionSpec and the Trend have been created, they are passed to the Template for conversion into DSyntS. The Template in this case is a VerbTemplate, which realizes the action as a main clause and the Trend as an adverb or adjective. The Template first converts the ActionSpec into the DSyntS shown in Figure 5.5. It then creates an adverb from the Trend object and attaches that to the generated DSyntS to create the structure in figure 5.6.

A similar process creates DSyntS fragments for each of the other nodes in the tree. Finally, the aggregator must combine these fragments into a single DSyntS; the BasicAggregator does this in the simplest possible way, by putting "and" between each of them. The DSyntS for the final caption is shown in Figure 5.7.

```
SPEND [ ] (
  I QUEBEC [ ]
  III healthcare [ class:common_noun article:no-art ]
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
)))
```

"Quebec spends on healthcare between 1990 and 1998."

Figure 5.5: DSyntS fragment for "Quebec" action

```
SPEND [ ] (
  I QUEBEC [ ]
  III healthcare [ class:common_noun article:no-art ]
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
  ))
  ATTR MORE1 [ rheme:+ ]
)
```

"Quebec spends more on healthcare between 1990 and 1998."

Figure 5.6: DSyntS fragment for "Quebec" action (Trend added)

5.2 Example 2: Grouping by parity

Obviously, there is a great deal of redundancy in the caption generated in Example 1; this redundancy could be removed by judicious grouping of the nodes in the tree. This involves choosing a different type of aggregator, a ParityAggregator. For this example, the data is the same; only the preamble of the input file has changed, as shown in Figure 5.8. Notice that the only difference between this one and Example 1 is in the choice of aggregator.

The initial tree for Example 2 is the same as the one for the previous example (see Figure 5.3); however, this time, when the aggregator is run, it reshapes the tree by sorting the nodes by the amount of change and then grouping nodes with like parity (increase, decrease, or no change). The result of this grouping is shown in Figure 5.9.

Now that the tree has been revised, the next step is to generate the DSyntS from each subtree. For the subtrees with one element, the process is exactly as in the previous example; the

```
SPEND [ ] (
  I  QUEBEC [ ]
  III  healthcare [ article:no-art class:common_noun ]
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
  )))
  ATTR  MORE1 [ rheme:+ ]
  COORD  AND2 [ ] (
    II  SPEND [ ] (
      I  ALBERTA [ ]
      III  healthcare [ article:no-art class:common_noun ]
      ATTR  BETWEEN1 [ ] (
        II  1990 [ class:numeral ] (
          COORD  AND2 [ ] (
            II  1998 [ class:numeral ]
      )))
      ATTR  LESS1 [ rheme:+ ]
      COORD  AND2 [ ] (
        II  SPEND [ ] (
          I  BRITISH_COLUMBIA [ ]
          III  healthcare [ article:no-art class:common_noun ]
          ATTR  BETWEEN1 [ ] (
            II  1990 [ class:numeral ] (
              COORD  AND2 [ ] (
                II  1998 [ class:numeral ]
          )))
          ATTR  MORE1 [ rheme:+ ]
          COORD  AND2 [ ] (
            II  SPEND [ ] (
              I  ONTARIO [ ]
              III  healthcare [ article:no-art class:common_noun ]
              ATTR  BETWEEN1 [ ] (
                II  1990 [ class:numeral ] (
                  COORD  AND2 [ ] (
                    II  1998 [ class:numeral ]
              )))
              ATTR  THE_SAME [ rheme:+ ]
              COORD  AND2 [ ] (
                II  SPEND [ ] (
                  I  MARITIMES [ ]
                  III  healthcare [ article:no-art class:common_noun ]
                  ATTR  BETWEEN1 [ ] (
                    II  1990 [ class:numeral ] (
                      COORD  AND2 [ ] (
                        II  1998 [ class:numeral ]
                  )))
                  ATTR  MORE1 [ rheme:+ ]
)))))))))
```

"Quebec spends more on healthcare between 1990 and 1998, Alberta spends less on healthcare between 1990 and 1998, British Columbia spends more on healthcare between 1990 and 1998, Ontario spends the same on healthcare between 1990 and 1998 and the Maritimes spend more on healthcare between 1990 and 1998."

Figure 5.7: Final DSyntS for Example 1

```
template: VerbTemplate
extractor: IncreaseDecreaseKey
aggregator: ParityAggregator
context: spend DirectObject( healthcare )
```

Figure 5.8: Input file preamble for Example 2

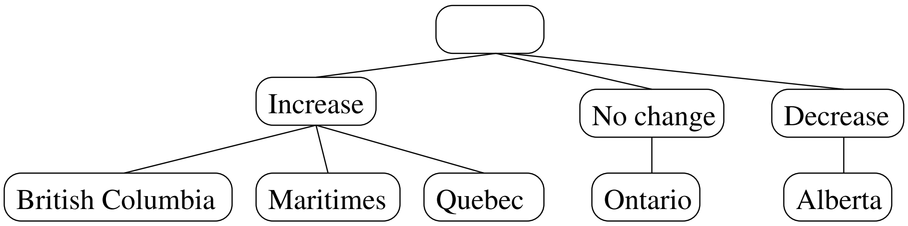


Figure 5.9: Revised tree for Example 2

interesting subtree is the "Increase" tree with its three leaves.

Since the tree was grouped by a ParityAggregator, the system knows that all of the nodes in each subtree have the same Trend; it can therefore create a single ActionSpec for the whole subtree and apply the Trend to that. The ActionSpec for the "Increase" subtree is shown in figure 5.10. Notice that, since all three actions differ in only one field of the ActionSpec, it was possible to create a single specification to cover all three of them. This is not always possible; see Example 6 for a case in which the ActionSpecs cannot be combined.

This ActionSpec creates a corresponding DSyntS which has all three of the specified subjects, to which the Trend is then applied as in Example 1; the result of this process is shown in Figure 5.11.

Finally, the ParityAggregator combines the fragments from each of its subtrees into a final DSyntS, shown in Figure 5.12. Notice that ParityAggregator uses "but", rather than "and", to combine the DSyntS.

Action:	[spend]
Subject:	[British_Columbia], [Maritimes], [Quebec]
Object:	[healthcare]
Complements:	[between [1990, 1998]]

Figure 5.10: Action specification of "Increase" subtree

```
SPEND [ ] (
  I  BRITISH_COLUMBIA [ ] (
    COORD  AND2 [ ] (
      II  MARITIMES [ ] (
        COORD  AND2 [ ] (
          II  QUEBEC [ ]
  ))))
  ATTR  MORE1 [ rheme:+ ]
  III  healthcare [ class:common_noun article:no-art ]
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
))))
```

"British Columbia, the Maritimes and Quebec spend more on healthcare between 1990 and 1998"

Figure 5.11: DSyntS fragment for "Increase" subtree

5.3 Example 3: Changing the Template

Examples 1 and 2 both used a VerbTemplate; this Template realizes the ActionSpec as a verb and the Trend as an adjective or adverb. By contrast, NounTemplate realizes the action as a noun phrase and the Trend as a verb. Figure 5.13 shows the preamble of the input file for Example 3. The only difference between this input file and the file in Example 2 is in the Template.

The process of generating this caption proceeds in the same way as described in Example 2 until the time comes for the Template to create DSyntS fragments from the subtrees. This time, the ActionSpec for the "Increase" subtree (Figure 5.10) is realized as a noun phrase as shown in Figure 5.14. The Trend is then realized as a verb and the DSyntS generated from the ActionSpec is attached as the verb's subject, resulting in the structure shown in Figure 5.15.

The fragments from the three subtrees are combined using "but", as in the previous section, resulting in the DSyntS shown in Figure 5.16.

5.4 Example 4: Comparing values

All of the preceding examples mention all of the values in the input; however, a good caption generally concentrates on just an "interesting" subset of the values: for example, the highest or

```
SPEND [ ] (
  I  BRITISH_COLUMBIA [ ] (
    COORD  AND2 [ ] (
      II  MARITIMES [ ] (
        COORD  AND2 [ ] (
          II  QUEBEC [ ]
  ))))
  III  healthcare [ article:no-art class:common_noun ]
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
  )))
  ATTR  MORE1 [ rheme:+ ]
  COORD  BUT [ ] (
    II  SPEND [ ] (
      I  ONTARIO [ ]
      III  healthcare [ article:no-art class:common_noun ]
      ATTR  BETWEEN1 [ ] (
        II  1990 [ class:numeral ] (
          COORD  AND2 [ ] (
            II  1998 [ class:numeral ]
      )))
      ATTR  THE_SAME [ rheme:+ ]
      COORD  BUT [ ] (
        II  SPEND [ ] (
          I  ALBERTA [ ]
          III  healthcare [ article:no-art class:common_noun ]
          ATTR  BETWEEN1 [ ] (
            II  1990 [ class:numeral ] (
              COORD  AND2 [ ] (
                II  1998 [ class:numeral ]
          )))
          ATTR  LESS1 [ rheme:+ ]
)))))
```

"British Columbia, the Maritimes and Quebec spend more on healthcare between 1990 and 1998, Ontario spends the same on healthcare between 1990 and 1998 but Alberta spends less on healthcare between 1990 and 1998."

Figure 5.12: Final DSyntS for Example 2

```
template: NounTemplate
extractor: IncreaseDecreaseKey
aggregator: ParityAggregator
context: spend DirectObject( healthcare )
```

Figure 5.13: Input file preamble for Example 3

```
SPENDING [ ] (
  ATTR  BRITISH_COLUMBIAN1 [ ] (
    COORD  AND2 [ ] (
      II  MARITIME [ ] (
        COORD  AND2 [ ] (
          II  QUEBECOIS [ ]
  ))))
  III  healthcare [ class:common_noun article:no-art ]
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
))))
```

"British Columbian, Maritime, and Quebecois spending on healthcare between 1990 and 1998"

Figure 5.14: DSyntS fragment for "Increase" subtree—noun phrase

```
INCREASE1 [ ] (
  I  SPENDING [ ] (
    ATTR  BRITISH_COLUMBIAN1 [ ] (
      COORD  AND2 [ ] (
        II  MARITIME [ ] (
          COORD  AND2 [ ] (
            II  QUEBECOIS [ ]
    ))))
    III  healthcare [ class:common_noun article:no-art ]
    ATTR  BETWEEN1 [ ] (
      II  1990 [ class:numeral ] (
        COORD  AND2 [ ] (
          II  1998 [ class:numeral ]
)))))
```

"British Columbian, Maritime, and Quebecois spending between 1990 and 1998 on healthcare increases"

Figure 5.15: DSyntS fragment for "Increase" subtree—noun phrase (Trend added)

the lowest value. The ComparisonAggregator implements the rules described by Corio (1999) that specify how the characteristics of the input data should determine which aspects of it are mentioned in the output. The input file preamble for Example 4 is shown in Figure 5.17; it differs from the input in Example 3 only in the choice of aggregator.

Once again, the initial tree is the same as in the previous examples (Figure 5.3). Once the initial tree has been created, the aggregator—in this case, a ComparisonAggregator—takes over. This aggregator uses the values of the data to select which dataset(s) should appear in the final caption. Since the value for British Columbia, which has the highest increase, increases more

```
INCREASE1 [ ] (
  I  SPENDING [ ] (
    ATTR  BRITISH_COLUMBIAN1 [ ] (
      COORD  AND2 [ ] (
        II  MARITIME [ ] (
          COORD  AND2 [ ] (
            II  QUEBECOIS [ ]
    ))))
    III  healthcare [ article:no-art class:common_noun ]
    ATTR  BETWEEN1 [ ] (
      II  1990 [ class:numeral ] (
        COORD  AND2 [ ] (
          II  1998 [ class:numeral ]
  ))))
  COORD  BUT [ ] (
    II  REMAIN [ ] (
      ATTR  THE_SAME [ ]
      I  SPENDING [ ] (
        ATTR  ONTARIO [ ]
        III  healthcare [ article:no-art class:common_noun ]
        ATTR  BETWEEN1 [ ] (
          II  1990 [ class:numeral ] (
            COORD  AND2 [ ] (
              II  1998 [ class:numeral ]
      ))))
      COORD  BUT [ ] (
        II  DECREASE1 [ ] (
          I  SPENDING [ ] (
            ATTR  ALBERTAN1 [ ]
            III  healthcare [ article:no-art class:common_noun ]
            ATTR  BETWEEN1 [ ] (
              II  1990 [ class:numeral ] (
                COORD  AND2 [ ] (
                  II  1998 [ class:numeral ]
)))))))))
```

"British Columbian, Maritime and Quebecois spending between 1990 and 1998 on healthcare increases, Ontario spending between 1990 and 1998 on healthcare remains the same but Albertan spending between 1990 and 1998 on healthcare decreases."

Figure 5.16: Final DSyntS for Example 3

```
template: NounTemplate
extractor: IncreaseDecreaseKey
aggregator: ComparisonAggregator
context: spend DirectObject( healthcare )
```

Figure 5.17: Input file preamble for Example 4

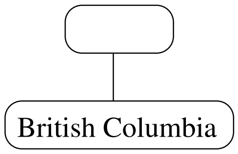


Figure 5.18: Revised tree for Example 4

```
INCREASE1 [ ] (
  ATTR  the_most [ class:adverb rheme:+ ]
  I  SPENDING [ ] (
    ATTR  BRITISH_COLUMBIAN1 [ ]
    III  healthcare [ article:no-art class:common_noun ]
    ATTR  BETWEEN1 [ ] (
      II  1990 [ class:numeral ] (
        COORD  AND2 [ ] (
          II  1998 [ class:numeral ]
)))))
```

“British Columbian spending between 1990 and 1998 on healthcare increases the most.”

Figure 5.19: Final DSyntS for Example 4

than 10% more than that for the next dataset (the Maritimes), the ComparisonAggregator decides that the B.C. dataset should be the only one in the final caption. The aggregator also stores the fact that the British Columbia dataset was chosen because it was the highest; this allows an an appropriate Trend to be selected later on. The revised tree is shown in Figure 5.18.

The other difference between Example 4 and the others is that a different Trend is chosen—one which reflects the fact that the remaining dataset not only increased, but had the highest increase. The final DSyntS generated by CAPUT on Example 4 is shown in Figure 5.19.

5.5 Example 5: Changing the MessageExtractor

In all of the preceding examples, it was the increase or decrease in data values that was used to sort the datasets and to choose an appropriate Trend to express. However, sometimes the user may want to concentrate on specific values rather than the overall change in value.

Figure 5.20 shows the input file preamble for Example 5. The extractor field has changed from an IncreaseDecreaseKey to a SingleValueKey; the parameter to the SingleValueKey indi-

```
template: NounTemplate
extractor: SingleValueKey( 0 )
aggregator: ComparisonAggregator
context: spend DirectObject( healthcare )
```

Figure 5.20: Input file preamble for Example 5

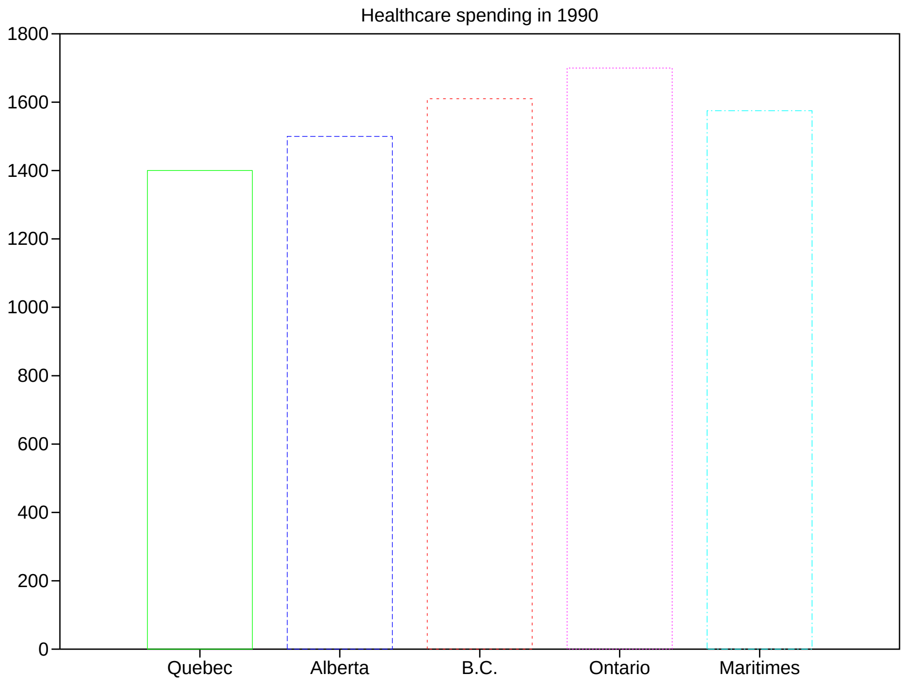


Figure 5.21: Graph of the sample data

cates that we are interested in the first value in each dataset. Figure 5.21 shows a graph of the data used in this comparison.

In this case, the values of all of the datasets are sufficiently close together that the ComparisonAggregator chooses to keep all of them in the final caption, shown in Figure 5.22.[2]

5.6 Example 6: Various syntactic changes

The preceding examples show the major changes that can be made in generating a caption. However, CAPUT can also make finer-grained choices, as the following example demonstrates.

[2]Notice the singular verb "has" in the final caption; unfortunately, RealPro does not consider subjects of this form to be plural.

```
HAVE1 [ ] (
  II  VALUE2 [ article:indef ] (
    ATTR  similar [ class:adjective ]
  )
  I  SPENDING [ ] (
    ATTR  ONTARIO [ ] (
      COORD  AND2 [ ] (
        II  BRITISH_COLUMBIAN1 [ ] (
          COORD  AND2 [ ] (
            II  MARITIME [ ] (
              COORD  AND2 [ ] (
                II  ALBERTAN1 [ ] (
                  COORD  AND2 [ ] (
                    II  QUEBECOIS [ ]
    ))))))))
    III  healthcare [ class:common_noun article:no-art ]
    ATTR  IN1 [ ] (
      II  1990 [ class:numeral ]
)))
```

"Ontario, British Columbian, Maritime, Albertan and Quebecois spending in 1990 on healthcare has a similar value."

Figure 5.22: Final DSyntS for Example 5

The input file for Example 6 is in Figure 5.23. There are several differences between this example and the others. First, the VerbTemplate is given a parameter, which will affect the form of the generated verb. Also, the context has been changed from one of spending to one dealing with food, and the objects (apple or orange) are attached to the individual datasets instead of to the caption specification as a whole.

The initial stages of processing this file follow those of Example 2 exactly, up to the point where the revised tree is created (Figure 5.9); the differences appear when generating DSyntS from that revised tree. Firstly, not all of the datasets in the subtree of increasing datasets can be combined into a single action specification, as they differ in too many fields; rather, they end up in two separate ActionSpecs, as shown in Figure 5.24. Also, the lexical entry for the verb *consume* specifies that it takes a direct object, as opposed to the prepositional object of *spend*. The object in the action specification is therefore realized as a direct object (a child of type "II" in the DSyntS), and the Trend as an adjective modifying the object. The ActionSpecs for the "Increase" subtree and the corresponding Trend object are realized as the DSyntS shown

```
template: VerbTemplate( aspect:cont )
extractor: IncreaseDecreaseKey
aggregator: ComparisonAggregator
context: food
-----
context: quebec DirectObject( orange )
1990    1400
1998    1450

context: alberta DirectObject( orange )
1990    1500
1998    1400

context: british_columbia DirectObject( apple )
1990    1610
1998    1900

context: ontario DirectObject( orange )
1990    1700
1998    1700

context: maritimes DirectObject( apple )
1990    1575
1998    1640
```

Figure 5.23: Input file for Example 6

Action:	[consume]
Subject:	[British Columbia], [Maritimes]
Object:	[apple]
Complements:	[between [1990, 1998]]

Action:	[consume]
Subject:	[Quebec]
Object:	[orange]
Complements:	[between [1990, 1998]]

Figure 5.24: Action specifications for Example 6

in Figure 5.25.

Another thing to note in the generated DSyntS in this case is that the specified attributes on the VerbTemplate have been passed on to the verbs in the DSyntS, so that the realization is in the present progressive rather than the simple present.

Similar actions take place to create DSyntS fragments in the other subtrees, and then the final DSyntS (Figure 5.26) is produced by combining these fragments using "but".[3]

[3]The use of "between" in this caption seems strange; CAPUT's handling of start and end dates could be improved.

```
CONSUME [ aspect:cont ] (
  I  BRITISH_COLUMBIA [ ] (
    COORD  AND2 [ ] (
      II  MARITIMES [ ]
  ))
  II  orange [ article:no-art class:common_noun ] (
    ATTR  MORE2 [ ]
  )
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
  )))
  COORD  AND2 [ ] (
    II  CONSUME [ aspect:cont ] (
      I  QUEBEC [ ]
      II  apple [ article:no-art class:common_noun ] (
        ATTR  MORE2 [ ]
      )
      ATTR  BETWEEN1 [ ] (
        II  1990 [ class:numeral ] (
          COORD  AND2 [ ] (
            II  1998 [ class:numeral ]
))))))
```

"British Columbia and the Maritimes are consuming more apple between 1990 and 1998 and Quebec is consuming more orange between 1990 and 1998."

Figure 5.25: DSyntS for the "Increase" subtree

.

5.7 Example 7: Computing totals

Example 6 lists the increase and decrease for each province and for each type of fruit separately. However, sometimes the desired content of a caption is the average or total value, rather than the individual values. This is where the TotalAggregator and AverageAggregator are useful.

The input file for Example 7 is shown in Figure 5.27. The data section of this file resembles that of Example 6, except that several of the provinces have been changed so that interesting totals can be computed. As well, the Template has been changed back to a NounTemplate, the MessageExtractor is a SingleValueKey, and the aggregator is now a TotalAggregator. Notice the argument to TotalAggregator; this indicates that it will compute a subtotal for each object that was consumed.

```
CONSUME [ aspect:cont ] (
  I  BRITISH_COLUMBIA [ ] (
    COORD  AND2 [ ] (
      II  MARITIMES [ ]
  ))
  II  orange [ article:no-art class:common_noun ] (
    ATTR  MORE2 [ ]
  )
  ATTR  BETWEEN1 [ ] (
    II  1990 [ class:numeral ] (
      COORD  AND2 [ ] (
        II  1998 [ class:numeral ]
  )))
  COORD  AND2 [ ] (
    II  CONSUME [ aspect:cont ] (
      I  QUEBEC [ ]
      II  apple [ article:no-art class:common_noun ] (
        ATTR  MORE2 [ ]
      )
      ATTR  BETWEEN1 [ ] (
        II  1990 [ class:numeral ] (
          COORD  AND2 [ ] (
            II  1998 [ class:numeral ]
  )))))
  COORD  BUT [ ] (
    II  CONSUME [ aspect:cont ] (
      I  ONTARIO [ ]
      II  apple [ article:no-art class:common_noun ] (
        ATTR  THE_SAME_AMOUNT_OF [ ]
      )
      ATTR  BETWEEN1 [ ] (
        II  1990 [ class:numeral ] (
          COORD  AND2 [ ] (
            II  1998 [ class:numeral ]
      )))
      COORD  BUT [ ] (
        II  CONSUME [ aspect:cont ] (
          I  ALBERTA [ ]
          II  apple [ article:no-art class:common_noun ] (
            ATTR  LESS2 [ ]
          )
          ATTR  BETWEEN1 [ ] (
            II  1990 [ class:numeral ] (
              COORD  AND2 [ ] (
                II  1998 [ class:numeral ]
))))))))
```

"British Columbia and the Maritimes are consuming more orange between 1990 and 1998 and Quebec is consuming more apple between 1990 and 1998, Ontario is consuming the same amount of apple between 1990 and 1998 but Alberta is consuming less apple between 1990 and 1998."

Figure 5.26: Final DSyntS for Example 6

```
template: NounTemplate
extractor: SingleValueKey
aggregator: TotalAggregator( object )
context: food
-----
context: ontario DirectObject( apple )
1990    1400
1998    1450

context: alberta DirectObject( apple )
1990    1500
1998    1400

context: ontario DirectObject( orange )
1990    1610
1998    1900

context: maritimes DirectObject( apple )
1990    1700
1998    1700

context: maritimes DirectObject( orange )
1990    1575
1998    1640
```

Figure 5.27: Input file for Example 7

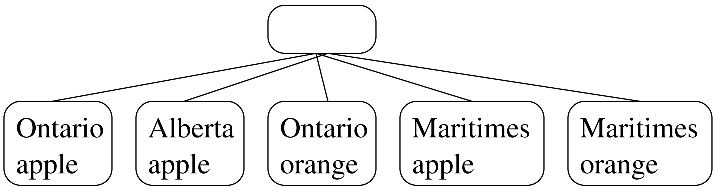


Figure 5.28: Initial tree for Example 7

The initial tree for this input file is shown in Figure 5.28. During the tree-modification process, a TotalAggregator behaves much like a ParityAggregator; that is, it creates new subtrees grouped by characteristics of the data. In this case, the TotalAggregator groups by object, so nodes with the same object are put together. The result of this grouping process is shown in Figure 5.29.

It is during the process of generating the DSyntS that the TotalAggregator actually computes the totals for each of its subtrees. We will consider only the "apple" subtree for this example; the same procedure takes place in the "orange" subtree as well.

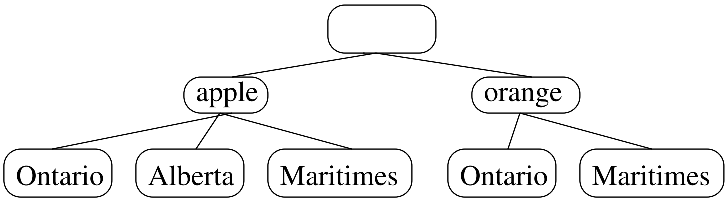


Figure 5.29: Revised tree for Example 7

Action:	[eat]
Subject:	
Object:	[apple]
Complement:	[in 1998]

Figure 5.30: Action specification of "apple" node

The ActionSpec for this subtree is shown in Figure 5.30. Notice that the "Subject" slot is empty; this is a result of the grouping by object that took place in the previous step. This ActionSpec is converted to the DSyntS shown in Figure 5.31.

The TotalAggregator then sums up the value in all of the datasets in its subtree; in this case, since a SingleValueKey was specified, it sums the final value from each dataset. It then creates a TotalTrend object to express the total value. The final DSyntS produced by this subtree is shown in Figure 5.32.

A similar process takes place to produce a DSyntS fragment from the "orange" datasets, and the two fragments are then combined using "and" to produce the final DSyntS shown in Figure 5.33. A similar process takes place when an AverageAggregator is used.

```
CONSUMPTION [ ] (
  II  apple [ article:no-art class:common_noun ]
  ATTR  IN1 [ ] (
    II  1998 [ class:numeral ]
))
```

"Consumption of apple in 1998"

Figure 5.31: DSyntS for the "apple" subtree

.

```
HAVE1 [ ] (
  II  VALUE2 [ article:indef ] (
    ATTR  TOTAL [ ]
    ATTR  OF1 [ ] (
      II  4550.0 [ class:numeral ]
  ))
  I  CONSUMPTION [ ] (
    II  apple [ article:no-art class:common_noun ]
    ATTR  IN1 [ ] (
      II  1998 [ class:numeral ]
)))
```

"Consumption of apple in 1998 has a total value of 4550.0."

Figure 5.32: DSyntS for the "apple" subtree

.

```
HAVE1 [ ] (
  II  VALUE2 [ article:indef ] (
    ATTR  TOTAL [ ]
    ATTR  OF1 [ ] (
      II  3540.0 [ class:numeral ]
  ))
  I  CONSUMPTION [ ] (
    II  orange [ article:no-art class:common_noun ]
    ATTR  IN1 [ ] (
      II  1998 [ class:numeral ]
  ))
  COORD  AND2 [ ] (
    II  HAVE1 [ ] (
      II  VALUE2 [ article:indef ] (
        ATTR  TOTAL [ ]
        ATTR  OF1 [ ] (
          II  4550.0 [ class:numeral ]
      ))
      I  CONSUMPTION [ ] (
        II  apple [ article:no-art class:common_noun ]
        ATTR  IN1 [ ] (
          II  1998 [ class:numeral ]
)))))
```

"Consumption of orange in 1998 has a total value of 3540.0 and consumption of apple in 1998 has a total value of 4550.0."

Figure 5.33: Final DSyntS for Example 7

.

Chapter 6

Conclusion

6.1 Summary

This thesis has presented the principles underlying the automated generation of captions for information graphics. A caption-generation system should address the generation tasks of content determination, discourse planning, sentence aggregation, lexicalization, referring expression generation, and linguistic realization. It should also have knowledge of various presentation techniques and of the domains about which it is generating, and should model both the user of the system and the eventual target audience of the presentation. A system to generate captions for graphics should be integrated with the system which is generating the graphics themselves, so that the final generated text and graphics complement each other and make appropriate reference across media.

CAPUT is a prototype system which attempts to follow these principles. It is implemented in Java, and it uses CoGenTex's RealPro text realizer for the text realization. CAPUT generates captions by creating an initial tree representation of its input, then reshaping that tree into a representation of the content of the final caption, and finally generating a sentence plan from the tree and sending it to RealPro for realization. CAPUT is composed of a number of abstract classes, with specific behaviour implemented in the subclasses of these classes. Captions which use new styles or which concentrate on new aspects of the data can be generated by creating new subclasses which implement the desired behaviour.

6.2 Contributions

6.2.1 Principles of caption generation

This thesis provides a description of the requirements for a system which is to generate text to accompany information graphics. Each task in a generalized text-generation architecture is examined from the perspective of caption generation. The factors that influence the processing at each stage are outlined.

Various possible approaches to implementing each of the generation tasks in a caption-generation system are provided; the methods used to address these tasks in existing systems are also outlined.

The other necessary components of a caption-generation system are presented: knowledge of the domain, knowledge of presentation techniques and of their effects on the audience, and a method of integrating the captions with the graphics which they are to accompany. Possible ways of implementing these components of a system are also suggested.

A distinction is made between text which summarizes quantitative data on its own and text which is designed to accompany information graphics. The content selected to appear in the text and the methods used to present it differ between these two cases, so different strategies must be employed in caption-generation systems than in systems which aim to generate stand-alone text.

6.2.2 CAPUT

CAPUT is a prototype implementation of a caption-generation system which attempts to address the principles outlined in Chapter 3. Although it does not directly address all of the listed requirements, it can be extended to address more of these with little difficulty.

It uses a tree-based method of content determination and discourse planning which is similar to the "topic trees" employed by GOSSIP and LFS. In this method, an initial unstructured tree of all of the information extracted from the raw data is modified top-down, sorting and grouping the datasets and potentially removing some of them, until the final tree represents the message structure of the text to be generated. This approach was taken because in the captions studied, some subset of the possible messages from the graph appeared, often sorted or grouped in some way; it was logical to mirror this structure in the process of producing these messages. Tree modification is a useful technique for this sort of generating domain, when coupled with sufficiently sophisticated rules guiding the processes of grouping the messages and producing text from the resulting tree.

The object-oriented nature of CAPUT is another strong point. All of the work in the system is performed by special-purpose subclasses of general abstract superclasses. All of the knowledge is represented in a procedural fashion, rather than a declarative one. In other words, the various components do not just specify what is to be done; they actually contain methods to perform the necessary actions. This means that potentially very complicated behaviour can be added to the system, and selected only when necessary by specifying the class which implements the desired behaviour. Any component in the system, down to those implementing the lexicon, can be subclassed if desired to obtain specific behaviour; this is potentially a very powerful feature for future development.

CAPUT is domain-independent, but capable of implementing very complex domain-specific behaviour simply by creating the necessary subclasses and then specifying them. At present, its use of domain knowledge is limited to selecting appropriate words to describe its data, but the domain model can be extended to influence all of the steps in the generation process.

6.3 Future directions

6.3.1 Short-term enhancements to CAPUT

CAPUT should be extended to produce a wider variety of caption types. One source of new types is the list provided by Corio (1999); possible extensions include:

- Explicitly mentioning variables which particularly interest the user—for example, the value in Quebec might be of special interest even if it is not the highest or lowest.
- Rather than concentrating only on an overall increase or decrease, using the data to choose between techniques such as mentioning only the last value, the trend of the data as a whole, or a trend in a sub-interval of time.
- Detecting correlation, non-correlation, convergence, and divergence of variables.

In addition to these schemata, a further corpus analysis of texts from other sources could reveal even more potential types of presentations.

In addition to providing a wider variety of captions, a useful end in itself, the process of implementing the classes necessary to produce these caption types may well reveal inadequacies in the current design of CAPUT; indeed, this has occurred several times during the development process up to this point. Any problems revealed during this process can then be fixed to make CAPUT more robust in the future.

Another possible short-term goal is to increase the number of domains about which CAPUT can generate captions. For the most part, this would be a simple task, requiring only additions to the lexical knowledge base or the specification of appropriate words in the input file. However, once again, testing CAPUT on a wider variety of domains may reveal inadequacies which, when rectified, will result in a stronger overall system which will be able to deal with even more domains.

6.3.2 Longer-term goals

The work listed in the previous section involves only short-term enhancements to make CAPUT more robust and general-purpose; however, there are also a number of potential longer-term enhancements that lead to other possible areas of research. The following are some of the possible future research areas in which CAPUT could play a part. Note that many of them interact with one another.

Use of a user model to select presentation techniques

As it currently stands, CAPUT has no knowledge itself of the rhetorical impact of various presentation techniques; all it knows is the techniques themselves. Other systems such as PostGraphe or AutoBrief allow the user of the system to express goals or intentions, such as *comparison* or *presentation*, which are then transformed into specific schemata for the generated text. Such integration would be valuable in CAPUT as well.

Existing research and guidelines on the selection of presentation techniques to produce an appropriate effect should be used to help in creating appropriate guidelines for the selection among textual presentation techniques.

In addition to rules governing the selection of a presentation technique, the set of goals which the user can specify should also be determined. A starting point might be Zelazny's "intentions", which are also used in PostGraphe/SelTex. However, these intentions govern the selection of graphical techniques; when text is to be generated, additional intentions might also be necessary to capture the full set of possible presentation goals.

There are two possible ways to integrate such a method of choosing between presentation techniques into CAPUT. First, the mapping between intended effect and presentation technique could be "hard-coded" into CAPUT, so that it selects an appropriate type of Aggregator and other objects to use in its generation. An alternative is a pre-processing step, in which some other program reads the user's goals and creates the appropriate input to CAPUT to produce the required sort of output.

Use of domain knowledge

CAPUT's use of domain knowledge in the current implementation is limited to using appropriate words to talk about the quantity being measured in the data—for example, "Canadian watermelon consumption". However, if the domain representation contains more complex information, much more interesting things can be done with that information.

The representation of a domain might include typical words and phrases used to talk about it in a variety of situations. For example, the phrasal lexicon for the stock market domain of Kukich (1983), includes phrases such as "mixed", "heavy trading", and other stock-specific terms. The lexicon used in FOG (Goldberg, Drieger and Kittredge, 1994), on the other hand, contains entries like "winds diminishing to light" and other phrases typically used in weather reports.

But beyond even specifying the terms to talk about a particular domain, the specification for a domain could also include rules about what features should be considered "interesting" and should appear in text referring to that domain. This information could describe, for example, the thresholds beyond which an increase or decrease is significant, or could specify other particular patterns in the data which are worthy of mention.

Integrating such domain knowledge into CAPUT requires multiple steps. First, the appropriate information should be obtained, either from consultation with domain experts or from a careful examination of typical texts from the domain—or, ideally, both. Then CAPUT will have to be extended to represent this knowledge in a way which can easily be used during the generation process. A possible method of doing this is to produce domain-specific subclasses of the important classes used during the process, but other representations may also be developed.

Automatic selection of presentation techniques

As noted above, CAPUT does not do any selection of presentation techniques on its own, but instead relies on the user or some external system to select the appropriate techniques. In future, the process of choosing among presentation techniques should be made more automatic.

The current system requires the user to specify precisely the technique to use and the particular aspects of the data to concentrate on. On the other extreme, the user could specify nothing at all about the desired features of the caption; it would then be up CAPUT to choose an appropriate presentation technique from its library of techniques.

If the user of the system gives no guidance at all, then CAPUT could perform a variety of tests in an attempt to determine which aspects of the data are the most salient or important; that is, whether the absolute increase of a single item is “more interesting” than the correlation or non-correlation of several of the variables, or some other characteristic.

To make such a choice automatically requires that the system be able to rate the salience of various characteristics of the data so that it can pick the one which is most worthy of note. This is a complex task, and one which certainly varies depending on the domain from which the data is drawn. For example, the different types of trend could be given varying weights depending on their nature and the magnitude or other measurements, and these weights would determine what appears in the final caption.

Although this enhancement was motivated by a scenario in which the user gives the system no guidance as to what presentation technique to choose, a similar method could be used with some user participation. For example, if a scheme of weights is used to select a technique, then the system user could specify revised weights to tailor the caption to a particular situation.

Integration with graphics generation

While CAPUT was designed to produce texts which can accompany information graphics, no work was done here on the generation of such graphics. A future application of CAPUT is to integrate it with a graphics generator in order to produce well-integrated presentations.

Before any media-specific generation can be done, the various components of the message must be allocated among the available media. This will ensure that each medium expresses the facts that it is best at; for example, the text should contain only the highlights of the message, while the graphic can provide support and background.

Once generation begins, the graphics- and text-generation systems should as a minimum use the same underlying representations so that the choices they make are compatible. Either both could take the same input, or there could be an overall controlling system which translates the user's input into specific goals for the two independent generation systems (as in AutoBrief).

For even more integrated results, the two systems should be able to communicate with each other; this will ensure that the caption complements the graphic without being redundant. If some important aspect of the data is already adequately expressed by the choices made in one medium, then the generator for the other medium could ignore that aspect to concentrate on others. This inter-generator communication can also allow the text generator to produce captions which explicitly refer to particular aspects of the graphic (as in SAGE), either because of their interest or because the particular graphical technique used is easy to misunderstand.

Text to replace graphics

CAPUT is a system designed to generate text to accompany graphics; a number of the previous systems generated only text with no consideration of graphics. A slightly different take on the same issue is to generate text to *replace* a graphic; for example, when the reader is visually impaired or when a set of pictures is to be indexed for searching.

The issues here vary greatly from those involved in generating text to accompany a graphic. The main difference is that, when the text and image are displayed together, the reader can be relied on to look at the image to fill in any details not mentioned in the text. When the graphic is to be replaced by words, on the other hand, the generated text is all that the user has available. The text must therefore describe all aspects of the image, rather than just highlighting some aspects of particular interest (as is often done with captions).

Every graphic in a well-designed document is there for a reason, whether it simply enhances the desired mood or it is crucial to the message the document is trying to convey. Determining the function of a specific image and how best to explain that function in words is not a trivial task. This process is the exact inverse of Zelazny's (1996) advice to presenters, in which he suggests that they first select a textual "message" and then produce a graphic to accompany it, with the message itself possibly not appearing explicitly in the final presentation.

A particular area where this sort of work is applicable is on the World Wide Web, where good style demands that authors provide alternative text for every image on a Web page so that it can be understood even by readers not using a graphical browser. Well-designed Web pages of this sort provide an excellent corpus for beginning a study of text used to replace graphics.

References

Arnold, K. and Gosling, J. (1997). *The Java Programming Language*. The Java Series. Addison-Wesley, second edition.

Boyd, S. (1999). *A Signal Processing Approach to Generating Natural Language Reports from Time Series*. PhD thesis, Department of Computing, Division of Information and Communication Sciences, Macquarie University.

Carcagno, D. and Iordanskaja, L. (1993). Content determination and text structuring: Two interrelated processes. In Horacek, H. and Zock, M., editors, *New Concepts in Natural Language Generation*, Communication in Artificial Intelligence Series, pages 10–26. Pinter Publishers.

Corio, M. (1999). *Sélection de l'information pour la génération de texte associé à un graphique statistique*. Master's thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal. Available online at `http://www.iro.umontreal.ca/~scriptum/CorioMsc.doc`.

Elhadad, M. and Robin, J. (1996). An overview of SURGE: A reusable comprehensive syntactic realization component. Technical Report 96-03, Department of Mathematics and Computer Science, Ben Gurion University, Beer Sheva, Israel.

Fasciano, M. (1996). *Génération intégrée de textes et de graphiques statistiques*. PhD thesis, Département d'informatique et de recherche opérationnelle, Université de

Montréal. Available online at `http://www.iro.umontreal.ca/~fasciano/recherche/postgraphe/these.pdf`.

Fidelity Investments (1998). History shows that stock markets have always rebounded. *Fidelity Directions*, 4(2):2. Available online at `http://www.fidelity.ca/english/library/directions/dmsm98b.html`.

Goldberg, E., Drieger, N., and Kittredge, R. I. (1994). Using natural language processing to produce weather forecasts. *IEEE Expert*, April:45–53.

Green, N., Carenini, G., Kerpedjiev, S., Roth, S., and Moore, J. (1998a). A media-independent content language for integrated text and graphics generation. In *Proceedings of the COLING-ACL '98 Workshop on Content Visualization and Intermedia Representations (CVIR '98)*, Montréal, Quebec, Canada. Available online at `http://www.cs.cmu.edu/~sage/Papers/cvir98.html`.

Green, N., Kerpedjiev, S., Roth, S. F., Carenini, G., and Moore, J. (1998b). Generating visual arguments: a media-independent approach. In *Proceedings of the AAAI-98 Workshop on Representations for Multi-modal Human-Computer Interaction*, Madison, Wisconsin. Available online at `http://www.cs.cmu.edu/~sage/Papers/AAAI98.html`.

Iordanskaja, L., Kim, M., Kittredge, R., Lavoie, B., and Polguère, A. (1992). Generation of extended bilingual statistical reports. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING-92)*, volume 3, pages 1019–1023, Nantes.

Kerpedjiev, S., Carenini, G., Green, N., Moore, J., and Roth, S. (1998). Saying it in graphics: from intentions to visualizations. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '98)*, pages 97–101, Research Triangle Park, NC. Available online at `http://www.cs.cmu.edu/~sage/Papers/sayit.html`.

Kerpedjiev, S., Carenini, G., Roth, S. F., and Moore, J. D. (1997). Integrating planning and task-based design for multimedia presentation. In *Proceedings of the International Conference* *on Intelligent User Interfaces (IUI '97)*, pages 145–152. Available online at `http://www.cs.cmu.edu/~sage/Papers/IUI-97/IUI-97.html`.

Kosslyn, S. (1994). *Elements of Graph Design*. W.H. Freeman, New York.

Kukich, K. (1983). Design and implementation of a knowledge-based report generator. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 145–150.

Lavoie, B. and Rambow, O. (1997). A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, pages 265–268, Washington, DC. Available online at `http://www.cogentex.com/papers/realpro-anlp97.ps`.

Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2):110–141.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.

Mittal, V. O., Moore, J. D., Carenini, G., and Roth, S. (1998). Describing complex charts in natural language: A caption generation system. *Computational Linguistics*, 24(3):431–468. Available online at `http://www.cs.cmu.edu/~sage/Papers/comchar.html`.

Norris, M. J. (1998). Canada's aboriginal languages. *Canadian Social Trends*, pages 8–16. Available online at `http://www.statcan.ca/english/ads/11-008-XIE/aborige.pdf`.

Reiter, E. and Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87. Available online at `http://www.csd.abdn.ac.uk/~ereiter/papers/jnle97.ps.gz`.

Robin, J. and McKeown, K. (1996). Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence*, 85:135–179. Available online at ftp://ftp.di.ufpe.br/pub/users/jr/aij.ps.gz.

Roth, S. F. and Hefley, W. E. (1993). Intelligent multimedia presentation systems: Research and principles. In Maybury, M., editor, *Intelligent Multimedia Interfaces*, pages 13–58. AAAI Press, Menlo Park, CA.

Roth, S. F., Mattis, J., and Mesnard, X. (1991). Graphics and natural language as components of automatic explanation. In Sullivan, J. W. and Tyler, S. W., editors, *Intelligent User Interfaces*, ACM Press Frontier Series, pages 207–239. Addison-Wesley.

Tufte, E. R. (1983). *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT.

Tufte, E. R. (1990). *Envisioning Information*. Graphics Press, Cheshire, CT.

USA Today (1999). Monster cookies. *USA Today*, page 1D. Available online at http://www.usatoday.com/snapshot/life/lsnap113.htm.

Zelazny, G. (1996). *Say It With Charts: The Executive's Guide to Visual Communication*. McGraw-Hill, New York.

Appendix A

Hierarchy of classes in CAPUT

The following is the hierarchy of the Java classes in the current implementation of CAPUT.

- class mef.thesis.**ActionSpec**
- class mef.thesis.**BasicObject**
  - class mef.thesis.**Aggregator**
    - class mef.thesis.**AverageAggregator**
    - class mef.thesis.**BasicAggregator**
    - class mef.thesis.**ComparisonAggregator**
    - class mef.thesis.**ParityAggregator**
    - class mef.thesis.**TotalAggregator**
  - class mef.thesis.**Field**
    - class mef.thesis.**Action**
    - class mef.thesis.**DirectObject**
    - class mef.thesis.**Subject**
  - class mef.thesis.**MessageExtractor**
    - class mef.thesis.**IncreaseDecreaseKey**
      - class mef.thesis.**PercentageKey**
    - class mef.thesis.**SingleValueKey**
  - class mef.thesis.**Template**
    - class mef.thesis.**NounTemplate**
    - class mef.thesis.**VerbTemplate**
- class mef.thesis.**Complement**
- class mef.thesis.**Contexts**

- class mef.thesis.**Dataset**
- class mef.thesis.**Input**
- class mef.thesis.**Main**
- class mef.thesis.**Point**
- class java.lang.Throwable
  - class java.lang.Exception
    - class java.lang.RuntimeException
      - class mef.thesis.**IncompatibleException**
- class mef.thesis.**TreeNode**
  - class mef.thesis.**LeafNode**
  - class mef.thesis.**ParentNode**
- class mef.thesis.**Trend**
  - class mef.thesis.**AverageTrend**
  - class mef.thesis.**IncreaseDecreaseTrend**
    - class mef.thesis.**IncreaseDecreaseCompareTrend**
      - class mef.thesis.**PercentageCompareTrend**
    - class mef.thesis.**PercentageTrend**
  - class mef.thesis.**SingleValueTrend**
  - class mef.thesis.**TotalTrend**