Applied Text Generation*

Owen Rambow
University of Pennsylvania
Department of CIS
Philadelphia, PA 19104
rambow@linc.cis.upenn.edu

Tanya Korelsky
CoGenTex, Inc.
105 Lenox Road
Ithaca, NY 14850
tanya@cogentex.com

1 Introduction

This paper presents the Joyce system as an example of a fully-implemented, application-oriented text generation system. Joyce covers the whole range of tasks associated with text generation, from content selection to morphological processing. It was developped as part of the interface of the software design environment Ulysses. The following design goals were set for it:

- The generated text must be of sufficiently high quality so that the user community of the underlying application accepts it as part of the documentation of software designs.
- The generation must be fast enough so that the system can be used as a tool during the design process.
- The system must be adaptable to new needs as they arise during further development of the underlying system, and it must be portable to completely new applications.

While we were able to exploit existing research for many of the design issues, it turned out that we needed to develop our own approach to text planning (Rambow 1990).

This paper will present the system and attempt to show how these design objectives led to particular design decisions. The structure of the paper is as follows. In Section 2, we will present the underlying application and give examples of the output of the System. In Section 3, we will discuss the overall structure of Joyce. We then discuss the three main components in turn: the text planner in Section 4, the sentence planner in Section 5 and the realizer in Section 6. We will discuss the text planner in some detail since it represents a new approach to the problem. Section 7 traces the generation of a short text. In Section 8, we address the problem of portability, and wind up by discussing some shortcomings of Joyce in the conclusion.

*Research on the original Joyce system (described in this paper) was supported by the AFSC at Rome Laboratory under grant no. F30602-85-C-0098 to Odyssey Research Associates. A successor system to Joyce has been under development at CoGenTex since early 1991. We would like to thank Richard Kittredge, Robert Rubinoff and two anonymous reviewers for helpful comments on earlier versions of this paper.

2 The Joyce System in the Ulysses User Interface

The Joyce text generation system was developped as part of the software design environment Ulysses (Korelsky and Ulysses Staff 1988; Rosenthal et al 1988) Ulysses includes a graphical environment for the design of secure, distributed software systems. The user manipulates icons that symbolize components (boxes) data ports of components (circles) and data flow between ports (arrows). Additional information, principally about the security level of the components and ports, is entered through menus. The design proceeds hierarchically by top-down structural decomposition.

As a tool in the user interface, Joyce generates two different types of texts about software designs:

- It generates annotations of the design which are intended to serve as part of the system documentation during and after the design process. A short and a long version of these texts are available. The long version can be several paragraphs long. The texts usefully complement the graphical representations since the graphical representation can show only one level in the structural decomposition, and since the additional information that is available about each component (in particular, security level) cannot be displayed graphically.
- It is used to explain the result of the application of a heuristic security design tool, the "flow analyzer"

The texts Joyce generates are specifically conceived of as written texts. The text output is integrated into the graphical environment in such a way that much of the same interactive functionality is available either through the text output window or through the graphical interface window. For example, if a designer reads the design annotation generated by Joyce and notices that the security level of a component has been entered wrong, then the error can be corrected by clicking at the name of the component in the text window and accessing the appropriate menu.

As an example of the output of Joyce, consider the text in Figure 2. It is an annotation of the component "Host" The top level decomposition of this component is shown in Figure 1. The text annotates the software design by describing its structure and interpreting it in terms of

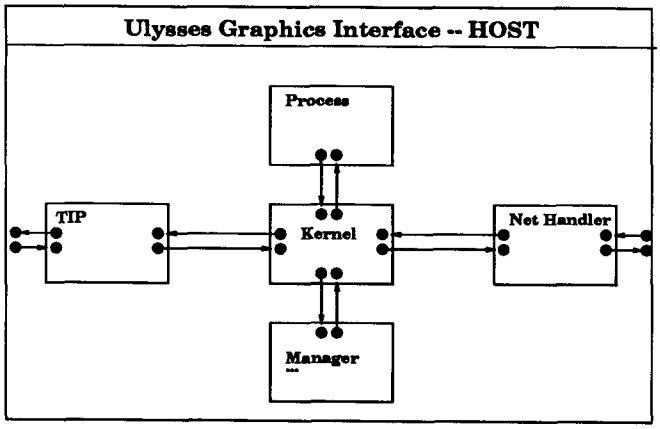

Figure 1: The HOST Graphical Representation

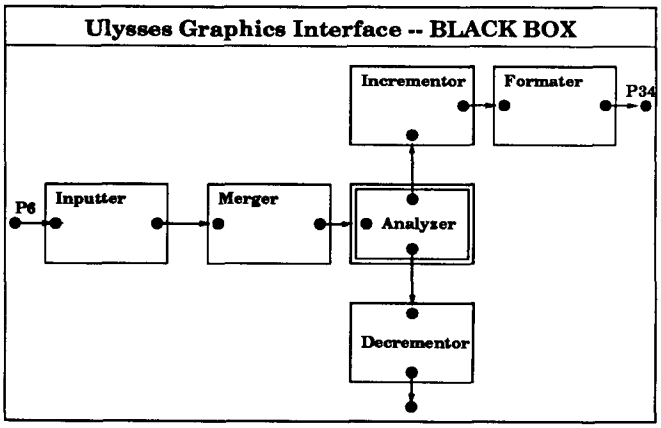

Figure 3: The BLACK BOX Graphical Representation

its security characteristics. The text in Figure 4 is generated by Joyce in order to report the results of the flow analyzer; the graphical representation of the underlying system can be seen in Figure 3. Note that the structures of the two texts are quite different: while the Host text is structured according to more abstract categories such as design structure and functionality, the Black Box text follows the path of the insecure flow through the component.

Joyce has been fully implemented in Common Lisp, and runs on the Symbolics Lisp Machine and on Sun workstations. A successor version has been ported to the Apple MacIntosh.

HOST: General Structure and Security Features

The multilevel Host is a complex component of the Station. It contains a Kernel, a TIP, a Process, a Net Handler and a group of Managers. The Process, the TIP, the Managers and the Net Handler communicate only through the Kernel. The manifestly secure Process and the Managers perform auxiliary functions. The Process is low-level. The TIP serves as interface to a User; the Net Handler handles communication with a Net. The security statuses of the TIP, the Managers and the Net Handler have not yet been specified.

The Kernel is a complex component. Its security status has not yet been specified. The Kernel contains a Message Switch, an Address Register and a Locator. The Address Register, the Locator and the Message Switch communicate directly with each other. The low-level Address Register and the multilevel Locator are data-bases. The Message Switch handles communication with the TIP, the Process, the Managers and the Net Handler. The security status of the Message Switch has not yet been specified.

Figure 2: The HOST Text

BLACK BOX: INSECURE FLOW

In the Black Box an insecure flow occurs. Classified information enters the Black Box through P6. It is passed through the Inputter to the Merger, which may upgrade it to top-secret. The Merger passes it to the Analyzer, which has been assumed secure. The Analyzer downgrades it to secret. It passes it through the Incrementor to the Formater, which downgrades it when a classified corrected reading leaves through P34.

Figure 4: The BLACK BOX Text

3 The Structure of Joyce

Joyce consists of three separate modules, which perform distinct tasks and access their own knowledge bases (Figure 5).

1. The *text planner* accesses the domain representation and produces a list of propositions, which represents both the content and the structure of the intended text. Each proposition is expressed in a language-independent, conceptual frame-like formalism. It encodes a minimal amount of information, but can be realized as an independent sentence if necessary. The text planner draws on domain communication knowledge expressed in a high-level schema language (see Section 4).
2. The *sentence planner* takes the list of propositions and determines how to express them in natural language. This task includes choosing lexicalizations and a syntactic structure for each proposition, and assembling these lexico-syntactic structures, called Deep Syntactic Representation or DSyntR, into larger sentences. It draws on knowledge captured in the conceptual/English dictionary.
3. The *linguistic realizer* takes the syntactic structures and produces surface sentences. It draws on syntactic and morphological knowledge, expressed in the English lexicon.

Usually, the different tasks of text generation are divided among two modules (planning and realization),

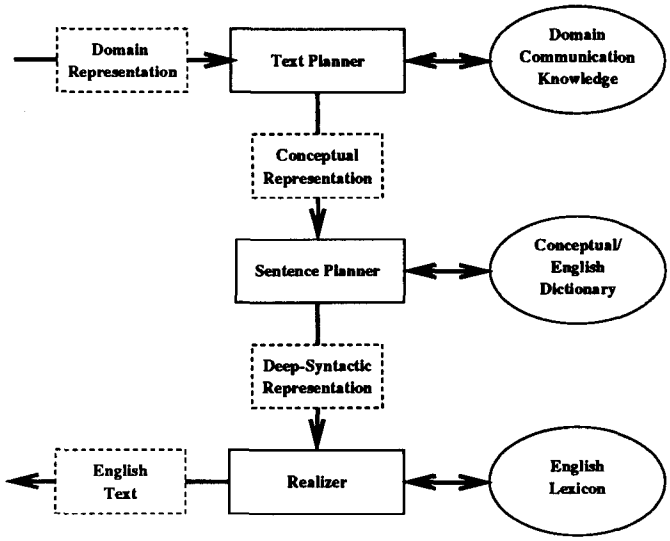

Figure 5: The Structure of Joyce

rather than three. However, there is a certain amount of disagreement about where the line between the two is to be drawn. For example, McKeown's TEXT (McKeown 1985) performs the tasks that Joyce classifies as sentence planning as part of the realization process, whereas Meteer's SPOKESMAN (Meteer 1989) classifies them as part of text planning. (See (Meteer 1990, p.23sq) for a useful summary of the terminological issues[1].) In this paper, "text planning" will always be used in the narrow sense of "content selection and organization". The architecture of Joyce is directly influenced by that of the SEMSYN system (Rösner 1987; Rösner 1988). Rösner divides the realization component into two parts, the "generator kernel" and the "generator front end". This distinction is mirrored exactly by the distinction between sentence planning and realization in Joyce.

There are two main advantages to such a tripartite architecture, one conceptual and the other practical. Conceptually, the advantage is that linguistic planning tasks are clearly separated from the actual grammar, which comprises word order and morphological rules. These rules can be stated independently of the formulation of purely semantic rules that determine lexical and syntactic choices. This modularity makes the system more maintainable. The linguistic planning tasks should, however, be clearly separated from the textual planning tasks: while the linguistic planning tasks are language-dependent, the textual planning tasks appear not to be[2]. Thus, if multi-lingual generation is desired, text planning and sentence planning ought to be performed by distinct components.

On a more practical level, modularity in design and implementation can be exploited by parallel processing of independent modules. While the current implementations of Joyce do not allow for parallel execution, the incremental processing of parallel computing tasks on a serial machine is also advantageous, as is argued in the WIP project (Wahlster et al 1991; Harbusch et al 1991)[3]. Incrementality reduces the initial response time of the system (though not the overall processing time). This can be crucial if multi-paragraph text is to be generated by an interface tool. In the Joyce system, the text planner cedes control to the sentence planner as soon as the text planner has defined a proposition. Once the sentence planner has constructed the DSyntR of a complete sentence, it sends it to the realizer which generates the English sentence. Thus, the first sentence is output by Joyce shortly after the text generator is invoked; text continues to be output approximately at reading speed. The effect is that a user of the text generator has the impression that he or she never has to wait for the system to respond, even when it is generating lengthy texts.

Throughout the system, processing is message-driven in the sense of (McDonald et al 1987): control lies in the input, which is used to construct the next level of representation. There is no need for backtracking or feedback from one level of processing to an earlier one. As is argued by McDonald *et al.*, such an architecture contributes to processing efficiency.

We will now discuss the three modules of Joyce in more detail.

4 The Text Planner

Prior to the design of the text planning component of Joyce, several existing approaches were studied. Since the structure of the descriptive text (Figure 2) does not mirror the structure of the domain, Paris's "procedural strategy" (Paris and McKeown 1987) cannot be used in general. Hovy's RST-based planner (Hovy 1988) assumes that content selection has already been performed, contrary to the situation in the Ulysses application; furthermore, there are efficiency problems in a pure STRIPS-like planning paradigm. We therefore found McKeown's schema-based approach (McKeown 1985) to be the most promising. However, it turned out that general rhetorical schemas cannot adequately capture the structure of the intended texts. In (Kittredge et al 1991), we argue that planning certain types of texts – such as reports and descriptions – requires domain-specific knowledge about how to communicate in that domain. That knowledge we call "domain communication knowledge" (DCK). For example, in describing secure system designs

[1] Note that the tasks Meteer groups together as "Syntax" – choosing the syntactic structure and linearization – are inseparable only in certain syntactic representations. In Joyce, the Deep-Syntactic Representation encodes syntactic structure but not linear order (see Section 6 for details).

[2] We are not aware of any example in which different text plans (as defined here) are needed for different languages. The fact that functionally similar texts may display different structures in different cultures should not be confused with language-specific constraints on text structure.

[3] Incrementality *within* the realizer has little practical benefit when the realizer is reasonably fast; its study is mainly motivated by psycholinguistic considerations. Therefore, there was no attempt in Joyce to make the realizer incremental.

you must relate the security level of each component, but not, say, the number of ports or their security levels. Furthermore, the connectivity of components should be stated before their functionality. In the flow analyzer text, the security levels of the components need not be communicated at all, but if a component (other than the final component of the path) downgrades information, it must be stated whether and why the component is secure. This very precise knowledge about which domain information needs to communicated and in what order cannot simply be derived from general principles. We have also argued that in many existing text planning systems, such DCK has been encoded implicitly. In the interest of efficiency, modularity and portability we have decided to represent DCK explicitly in Joyce.

We have developed a "schema language" for easy representation of DCK, called DICKENS (DomaIn Communication Knowledge ENcoding Schemas). The schemas are similar in form to those used by McKeown. Basically, schemas can be seen as a description of text structure. The system, however, interprets each schema as a list of instructions. The instructions can be calls to other schemas, recursive calls to the same schema, or they can be one of a set of special commands provided by the schema language. One special command produces a specific proposition and sends it to the sentence planner. Other special commands support conditional branching and iteration. During execution, each schema is associated with a particular subset of the domain representation, which is called the focus (in the sense of McKeown's "global focus"). In the Ulysses application, the focus always corresponds to one component. There are special commands to shift the focus. In addition to the focus, which limits the domain representation from which information can be communicated, a theme can be set which determines information structure within individual propositions. The theme corresponds to McKeown's "local focus". As has been widely recognized, thematic structure affects issues such as grammatical voice at the linguistic level.

In addition, two further special commands were found to be necessary in order to perform text planning:

- A portion of the text plan can be edited. To do this, a schema is called, but any propositions that are created (by the schema or by any schema it calls) are not sent to the sentence planner. They are kept on a separate list in the order they are created. When the execution of the schema terminates, an editing function is applied to the list. The editing function can delete propositions, change their order, change their contents or create new ones. The choice of an editing function depends on the domain and on the particular requirements of the text. Further study is needed in order to determine the types of editing operations that can be made and to devise a high-level language to express them; the goal is to eventually establish a library of editing operations. Typical editing operations we have used include juxtaposing similar propositions or juxtaposing propositions with certain similar slots (typically, the agent slot). An example is given in Section 7.

  This type of revision is different from the revision discussed in (Gabriel 1988) and (Meteer 1991). In these systems, the *linguistic* specification of the target texts is revised. In Joyce, it is the text plan itself, i.e. the *pre-linguistic* representation of text content and structure, that is subject to revision.

- Schemas can post to a "blackboard", and check this blackboard for messages. This allows for additional control and communication between schemas which are called at different times during the text planning process and cannot communicate with each other directly.

Instead of being templates that limit the structure of the text to certain preconceived types, the schemas are now an explicit and compact representation of domain communication knowledge.

5 The Sentence Planner

The sentence planner combines all those planning tasks that are specific to the target language. It receives propositions from the text planner and sends the DSyntR of complete sentences to the realizer for processing. It has two main tasks: first, it chooses lexical and syntactic realizations by consulting the Conceptual/English dictionary; second, it determines sentence scope by merging the DSyntR of individual propositions. We will discuss each of these steps in turn.

The Conceptual/English dictionary is implemented as a set of procedures that operate on the propositions. Each proposition is mapped into the DSyntR of a clause (i.e., its root is a verb). Lexicalization can take pragmatic factors into account. It can also refer to a history of lexicalizations if lexical variation is desired. After a DSyntR has been constructed, certain syntactic paraphrase operations are performed if necessary, for example passivization if a grammatical object is the theme of the sentence, or if the subject is absent.

The second task of the sentence planner is to determine the scope of sentences. Combining the linguistic realization of propositions into larger sentences is a crucial issue because it increases the quality of the generated text. For example, *The low-level Address Register and the multilevel Locator are data-bases* (from the Host text in Figure 2) is significantly better than the four clauses from which it was formed: *The Address Register is a data-base. It is low-level. The Locator is a data-base. It is multilevel.* An informal study in which subjects were asked to revise a (grammatical) text containing only single-proposition sentences supported the claim that longer sentences are preferred over shorter ones whenever possible and reasonable.

The first question that arises is at what level propositions should be combined. To date, the issue of sentence scoping has always been dealt with at a pre-linguistic, conceptual level (e.g. (Dale 1988) or (Carcagno and Iordanskaja 1989)). However, different languages have different syntactic means of combining clauses; clause combining must refer to the specific linguistic resources of the target language. Therefore, in Joyce the task is performed by the sentence planner rather than the text

planner[4]. Joyce performs the following syntactic clause-combining operations: Relative clause formation, adjectival attachment (the process by which an adjective from a copula-construction is embedded in an NP), and conjunction. Conjunction includes multiple conjunctions of more than one clause, and may lead to elision of repeated sentence elements ("conjunction reduction"). For example, in the example quoted above, the lexeme *data base* occurs only once in the conjoined sentence.

The second question that arises is how clause combination should be restricted. We have identified stylistic and discourse constraints. The stylistic constraints are constraints against the sentence becoming too long (an upper bound on the number of clauses that can be combined into one sentence), and a constraint on recursive embedding of relative clauses. Discourse constraints are imposed by the structure of the text: clauses belonging to conceptually distinct text units should not be combined. The text planner can send a special message, called `conceptual-break`, to the sentence planner. It signals the beginning of a new textual unit. These special messages are triggered by appropriate indications in the DICKENS specifcation of the DCK.

The algorithm is as follows. The sentence planner maintains a "current" DSyntR. Each incoming proposition is translated into a DSyntR, which the sentence planner then attempts to merge with the current DSyntR. If none of the clause combination strategies work, or if stylistic heuristics interfere, or if the incoming proposition is a `conceptual-break`, the current DSyntR is sent to the realizer and the new DSyntR becomes the current one. The process of clause combination can be very easily modeled at the DSyntR level: relative clause formation and conjunction reduce to simple tree composition operations. (In the case of adjectival attachment only the adjective node is attached.) Issues such as word order in relative clauses, the morphological form of the complementizer, and conjunction reduction can be dealt with at further stages of processing.

6 The Linguistic Realizer

The linguistic component is based on Meaning-Text Theory (MTT) (Mel'čuk 1988), and is a reimplementation (in Lisp) of Polguère's Prolog implementation of a Meaning-Text model for English (Iordanskaja et al 1988; Iordanskaja et al 1991).

MTT defines three successive levels of representation. With each level of representation is associated a component which transforms the representation into the next higher level. Each component is implemented as a separate module in Joyce.

- The Deep-Syntactic Representation (DSyntR) is a dependency grammar tree representing the syntactic relationships between the meaning-bearing words of a sentence. Sister nodes are unordered with respect to each other. The nodes are labelled with lexemes which are annotated with features. Numerical arc labels represent the syntactic arguments of the governing lexeme, while ATTR represents the attributive relation. An example is shown in Figure 6. Note that the function words *the*, *is*, *to* are not yet represented.

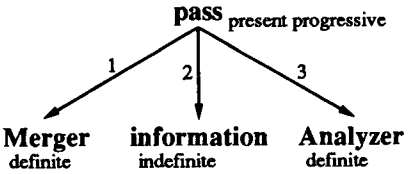

Figure 6: DSyntR of sentence *The Merger is passing information to the Analyzer*

- The Surface-Syntactic Representation (SSyntR) is also a dependency grammar representation, but it includes all lexemes of the final sentence. The transition between DSyntR and SSyntR is achieved by looking up function words in the English lexicon, and by expanding grammatical features such as verb tenses.
- The Deep Morphological Representation (DMorphR) is a linearization of the nodes of the SSyntR.
- The Surface Morphological Representation is in fact the written form of the English sentence. Morphological processing is done by a component closely based on SUTRA-S (Emele and Momma 1985).

While linguistic realizers based on other theories could have been used, this MTT-based approach offers the following advantages:

- The approach is based on an independently motivated linguistic theory. Much linguistic work has already been done in the MTT framework (for example (Mel'čuk and Pertsov 1987)).
- The modularization of different types of linguistic knowledge makes the grammar easier to maintain. Parallelism in computation could be exploited.
- The dependency grammar used to express the two syntactic levels of representation permits the separation of the semantically relevant issue of grammatical relations (e.g., subjecthood) from pragmatically relevant issues of surface word order (e.g., topicalization).

7 An Example

As an example, consider the sample text in Figure 4. It describes the occurrence of an insecure flow in component Black Box. The texts that explain insecure flow are generated by a set of eight schemas, one of which is shown in Figure 7. It is the first that is invoked.

Special commands are preceded by a colon; command not starting with a colon are calls to other schemas

[4]In Section 7, we discuss an example in which two propositions are merged by the text planner. The crucial point is that in that example, the two propositions are merged into a *single* proposition. Here, we are discussing cases in which *two distinct* propositions are linguistically realized in the same sentence.

```
(defschema flow-analysis-error
  :title "Insecure flow"
  :theme "information"
  :make-proposition (insecure-flow :location focus)
  :make-proposition (enter :agent (get-information)
                           :object focus
                           :location (entry-port focus))
  :make-proposition (id-security :agent (get-information)
                                 :value (get-level (entry-port focus)))
  conceptual-break
  :shift-focus-and-edit (next-component initial-follow-path #'merge-send-data))
```

Figure 7: The FLOW ANALYZER schema

The arguments to special commands immediately follow the command. The `:title` special command generates a title. Command `:theme` sets the initial theme of the paragraph, influencing issues such as passivization. Then follow three `:make-proposition` commands, which each produce one proposition. The first argument to `:make-proposition` is the class of the proposition. The slots are typically filled with pointers into the domain representation of the application program. `focus` is a pointer maintained by the text planner which refers to the global focus (currently the component Black Box, represented by pointer `#<COMPONENT Black Box>`), while `get-information` and `entry-port` are functions provided by the underlying application program. Not all arguments must be filled by a `:make-proposition` command; the sentence planner will choose lexical and syntactic realizations accordingly. The text planner sends an `insecure-flow` proposition to the sentence planner, which translates it into a DSyntR tree (which represents the clause *In the Black Box an insecure flow occurs*) and returns control to the text planner. The text planner then proceeds to the next `:make-proposition` command, and sends the proposition shown in Figure 8 to the sentence planner. When the sentence planner re-

```
ENTER
  AGENT     #<information>
  OBJECT    #<COMPONENT Black Box>
  LOCATION  #<PORT P9>
```

Figure 8: The ENTER proposition

ceives the `enter` proposition, it translates it into the DSyntR tree shown in Figure 9, which could be expressed as the clause *information enters the Black Box through P6*. Note that the choice of *enter* as verb is due to the fact that *information* is currently the theme; if *Black Box* were the theme, the choice would have been *receives*. The sentence planner then tries to combine the new DSyntR with the current one (which was derived from the previous proposition). This fails (since the two clauses have different verbs and different actants), so the current DSyntR is sent to the realizer, which prints out the first sentence. The new DSyntR

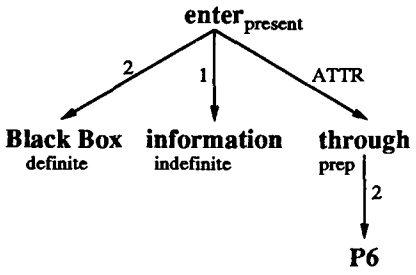

Figure 9: DSyntR of sentence *information enters the Black Box through P6*

becomes the current one. Control is returned to the text planner, which processes the third `:make-proposition` command and sends the appropriate proposition to the sentence planner. The sentence planner generates the clausal DSyntR tree shown in Figure 10 (*the information is classified*). It then attempts to combine the new

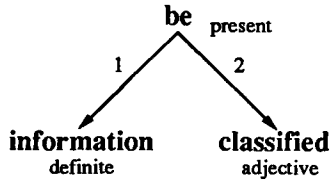

Figure 10: DSyntR of sentence *The information is classified*

clause with the "current DSyntR", first using the adjectival attachment strategy. This succeeds, yielding the tree shown in Figure 11. It then returns control to the text planner, since another clause could be merged with the current DSyntR. The text planner then calls schema conceptual-break. The only effect of this schema is to send a `conceptual-break` message to the sentence planner, which thereupon sends its current DSyntR to the realizer. The realizer prints out the surface sentence *Classified information enters the Black Box through P6.*

The last command of the schema first shifts the (global) focus to `next-component`, which is the next component traversed by the insecure flow. The second argument of the `:shift-focus-and-edit` command designates the next schema to be called. This com-

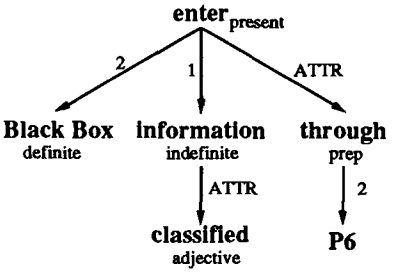

Figure 11: DSyntR of sentence *Classified information enters the Black Box through P6*

mand also initiates the editing process. All propositions that are generated as a result of this command are kept on a list rather than sent to the sentence planner. When the command has been executed, the list is edited by the function given as the third argument, `#'merge-send-data`. The effect of this function is to combine two successive **send** propositions into a single, new one, so that two clauses such as *the Analyzer sends the information to the Incrementor* and *the Incrementor sends the information to the Formater* yield *the Analyzer sends the information to the Formater through the Incrementor*. Note that this combination is not a linguistic one but a conceptual one, since it relies on facts about sending data in this domain, rather than on the syntax or lexical semantics about the verb *send*. It must therefore be performed by the text planner, and not the sentence planner.

8 Porting the System

Porting is an important way to evaluate complete applied text generation systems, since there is no canonical set of tasks that such a system must be able to perform and on which it can be tested. (Realization components, on the other hand, can be tested for their syntactic and perhaps lexical coverage.) Joyce was originally designed to generate only component descriptions (as in Figure 2). The "flow analyzer" heuristic tool was added later to the system, and the completely different type of text it required was a first successful test of Joyce and its text planner in particular.

The modular design of Joyce proved beneficial during the porting to the new application. The following conceptually well-defined tasks were required during the development of the "flow analyzer" application:

1. Since the flow analyzer is a new type of tool, no corpus of texts was available for study. Instead, sample texts were written by hand and critiqued by domain experts. The texts were then revised and resubmitted to the experts. The "ideal text" that emerged was then analyzed and the DCK needed to generate it expressed in terms of schemas. We interpret the cycle of writing, critiquing and revising as a process of DCK acquisition.

2. New classes of proposition were defined. These include **enter**, **upgrade** and **downgrade**. Some of the proposition classes from the earlier descriptive application could be reused, such as **send**.

3. The Conceptual/English dictionary was extended to account for the new proposition classes.

4. Several new lexical items were entered into the English lexicon. For example, the English lexeme *downgrade* subcategorizes for two nouns and a propositional phrase obligatorily headed by *to*.

Note that those parts of Joyce that deal with facts of English (including clause combination) needed no attention (other than updating the lexicon).

We are currently working on porting a successor of Joyce to several new applications, including the generation of project management reports. Initial results, including a prototype, are encouraging.

9 Conclusion

We are aware of several shortcomings of Joyce, which we will address in future versions of the system.

- While we have argued in (Kittredge et al 1991) that rhetoric cannot be the central guiding principle in text planning, it appears to play an important role as a constraint on possible text structures. Furthermore, it helps determine the use of connectives between rhetorically related clauses. Finally, it may determine when conceptual breaks occur in text structure which affect sentence scoping (Scott and de Souza 1990). We are currently investigating the option of augmenting the DCK schemas with rhetorical annotations.
- The current form of the Conceptual/English dictionary is not satisfactory, since the dictionary writer is too free in writing dictionary entries. For example, the dictionary could be used as a back door for the introduction of new content which the text planner was (for whatever reasons) unable to plan. Meteer discusses the same problem in McKeown's original TEXT system (Meteer 1990, p.35). An interface to the dictionary that is more restrictive is needed.
- While it is possible to set a theme in the text plan, thematic structure has not received sufficient attention. Rules of thematic progression (as implemented, for instance, in McKeown's TEXT) are not taken into consideration. Furthermore, clause combination is also sensitive to thematic structure (Kuno 1976; Derr and McKeown 1986; Iordanskaja 1989), which is currently not taken into account.

Despite these shortcomings, Joyce has proven to be a successful and useful tool in the Ulysses user interface. It has met the design objectives of speed and quality, and our experience in porting the text generator to new tasks and to new applications indicates that Joyce is a flexible system that can adapt to a variety of text generation tasks.

Bibliography

Carcagno, Denis and Iordanskaja, Lidija, 1989. Content Determination and Text Structuring in Gossip. In *Proceedings of the Second European Workshop on Text Generation*. Edinburgh.

Dale, Robert, 1988. *Generating Referring Expressions in a Domain of Objects and Processes*. PhD thesis, University of Edinburgh.

Derr, Marcia A. and McKeown, Kathleen R., 1986. Using Focus to Generate Complex and Simple Sentences. In *24th Meeting of the Association for Computational Linguistics (ACL'86)*, ACL, pages 319–326.

Emele, Martin and Momma, Stefan, 1985. *SUTRA-S: Erweiterungen eines Generator-Front-End für das SEMSYN-Projekt*. Technical Report, Universität Stuttgart.

Gabriel, Richard P., 1988. Deliberate Writing. In McDonald, David D. and Bolc, Leonard (editors), *Natural Language Generation Systems*, pages 1–46. Springer Verlag.

Harbusch, Karin; Finkler, Wolfgang; and Schauder, Anne, 1991. Incremental Syntax Generation with Tree Adjoining Grammars. In *Proceedings 4.Int. GI-Kongress Wissensbasierte Systeme*, GWAI. München.

Hovy, Eduard H., 1988. Planning coherent multisentential text. In *Proceedings of the 26th Annual Meeting*, ACL, pages 163–169. Buffalo.

Iordanskaja, Lidija, 1989. *Communicative Structure and Its Use During Text Generation*. Technical Report TR 15-10, Odyssey Research Associates, Ithaca, NY/Montréal, PQ.

Iordanskaja, Lidija; Kittredge, Richard; and Polguère, Alain, 1988. Implementing the Meaning-Text Model for Language Generation. Paper presented at COLING-88.

Iordanskaja, Lidija; Kittredge, Richard; and Polguère, Alain, 1991. Lexical Selection and Paraphrase in a Meaning-Text Generation Model. In Paris, Cecile; Swartout, William; and Mann, William (editors), *Natural Language Generation in Artifical Intelligence and Computational Linguistics*, pages 293–312. Kluwer Academic Publishers.

Kittredge, Richard; Korelsky, Tanya; and Rambow, Owen, 1991. On the Need for Domain Communication Knowledge. *Computational Intelligence* 7(4).

Kuno, S., 1976. Subject, theme and the speaker's empathy – a reexamination of relativization phenomena. In Li, Charles N. (editor), *Subject and Topic*, pages 417–444. Academic Press.

McDonald, David D.; Meteer (Vaughan), Marie W.; and Pustejovsky, James D., 1987. Factors contributing to efficiency in Natural Language Generation. In Kempen, Gerard (editor), *Natural Language Generation*, pages 159–181. Martinus Nijhoff Publishers.

McKeown, Kathleen, 1985. *Text Generation*. Cambridge University Press, Cambridge.

Mel'čuk, Igor A., 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, New York.

Mel'čuk, Igor A. and Pertsov, Nikolaj V., 1987. *Surface Syntax of English*. John Benjamins, Amsterdam/Philadelphia.

Meteer, Marie W., 1989. *The SPOKESMAN Natural Language Generation System*. Technical Report, BBN Systems and Technologies Corporation.

Meteer, Marie W., 1990. *The 'Generation Gap': The Problem of Expressibility in Text Planning*. Technical Report 7347, BBN Systems and Technologies Corporation.

Meteer, Marie W., 1991. The Implication of Revisions for Natural Language Generation. In Paris, Cecile; Swartout, William; and Mann, William (editors), *Natural Language Generation in Artifical Intelligence and Computational Linguistics*, pages 155–177. Kluwer Academic Publishers.

Paris, Cecile L. and McKeown, Kathleen R., 1987. Discourse Strategies for Describing Complex Physical Objects. In Kempen, Gerard (editor), *Natural Language Generation*, pages 97–115. Martinus Nijhoff Publishers.

Rambow, Owen, 1990. Domain Communication Knowledge. In *Proceedings of the Fifth International Workshop on Natural Language Generation*. Dawson, PA.

Rösner, Dietmar, 1987. The Automated News Agency SEMTEX – a Text Generator for German. In Kempen, G. (editor), *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pages 138–148. Kluwer Academic Publishers, Boston.

Rösner, Dietmar, 1988. The SEMSYN Generation System: Ingredients, Applications, Prospects. In *Proceedings of the Second Conference on Applied Natural Language Processing*, ACL. Austin.

Scott, Donia R. and de Souza, Clarisse Sieckenius, 1990. Getting the message across in RST-based Text Generation. In Dale, Robert; Mellish, Chris; and Zock, Michael (editors), *Current Research in Natural Langugae Generation*. Academic Press, London.

Wahlster, Wolfgang; Andre, Elisabeth; Graf, Winfried; and Rist, Thomas, 1991. WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation. In *Proceedings of the 5th Conference of the European Chapter*, ACL. Berlin.