Integrated Natural Language Generation Systems

Mark Kantrowitz Joseph Bates

April 1992

CMU-CS-92-107

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in *Aspects of Automated Natural Language Generation*, R. Dale, E. Hovy, D. Rösner and O. Stock (Eds.), LNAI Volume 587, Springer-Verlag, 1992.
Presented at the *6th International Workshop on Natural Language Generation*, Trento, Italy, April 1992.

This work was supported in part by Fujitsu Laboratories, Ltd., and in part by a National Science Foundation (NSF) Graduate Fellowship.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Fujitsu Laboratories or the NSF.

Keywords: Natural Language Generation, Planning, Uniform Control and Representation, Process Models of Language Production, Artificial Intelligence, Interactive Fiction, Virtual Reality, Oz

Abstract

Many existing natural language generation systems can be characterized according to their modularization as either *pipelined* or *interleaved*. In these *separated* systems, the generator is divided into several modules (e.g., planning and realization), with control and information passing between the modules during the generation process. This paper proposes a third type of generator, which we call *integrated*, that unifies the modules into a single mechanism. The mechanism uses a small set of orthogonal basic operations to produce planned and grammatical language output.

Integrated systems are conceptually attractive and may support generation of pragmatic effects more effectively than other systems. After discussing the advantages of the integrated approach, we summarize GLINDA, an integrated generator currently under development at Carnegie Mellon. GLINDA is the generator used for narration and intercharacter communication in the Oz Interactive Fiction and Virtual Reality Project.

1. Introduction

This paper distinguishes two classes of natural language generation (NLG) systems, which we call *separated* and *integrated*. A separated system divides the generator into several specialized modules, with each module having different representations and processes, corresponding to one or more hierarchically organized linguistic levels [6, 11]. Separated systems often include a *planning* component, which is responsible for selecting and organizing the conceptual content of the message to be expressed, and a *realization* component, which is responsible for executing the text plan and producing grammatical natural language output. These modules are also known by the names deep and surface generation, strategic and tactical generation, text planning and plan execution, message and form levels, functional and positional levels, and conceptual and grammatical levels. Some systems further subdivide the realization component into modules that parallel linguistic levels such as phonology, morphology, lexical selection, and phrase, clause, and sentence structure.

Besides using distinct representations and processing mechanisms in each module, separated systems often define sharply constrained module interfaces which restrict cross-level information flow. However, motivated in part by our application's demand for more flexible communication between levels (see also [12, 36]), we have tried to identify commonalities in the operations and representations used at each linguistic level and to generalize them into a simple framework that can be used for all aspects of generation. The resulting *integrated* generator has a single engine, with any modularization of the generator appearing solely in the organization of its rules.

In this paper we argue in favor of integrated systems, describing the motivation, characteristics, and benefits of such systems, and suggest that they may be feasible to build. Our initial architectural ideas are embodied in a program called GLINDA which is beginning to be used for narration and intercharacter communication in the Oz Interactive Fiction and Virtual Reality Project at Carnegie Mellon. The generation rules which drive GLINDA also are under active development.

1.1. Types of Natural Language Generators

The three types of generation systems are illustrated in Fig. 1.

Separated systems can be roughly divided into *pipelined* systems and *interleaved* systems [17, 19, 27]. In a pipelined generator, all text planning decisions must be completed before any surface realization may begin. The planning stage produces a fully specified and ordered message which is then translated into natural language text by the realization stage. The text planner is completely free of any linguistic knowledge and the realizer has only linguistic knowledge available to it. There is no communication from the realization module back to the planning module. Examples of such systems include McKeown's TEXT [28, 29] and Jacob's PHRED [20, 21].[1]

[1] Unless a researcher has published flow charts or descriptions of a generation algorithm, it may be difficult to decide whether a generator is pipelined, interleaved or integrated. It is our hope that this paper will popularize the use of this terminology for the description and discussion of NLG systems. We regret if we've inadvertently misclassified a generator.

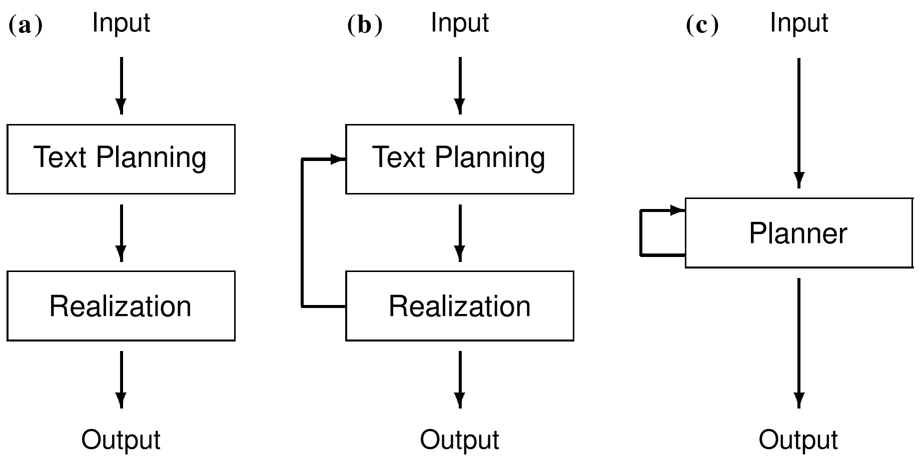

Figure 1: Generator Organizations: **(a)** Pipelined, **(b)** Interleaved, and **(c)** Integrated

Pipelined generators are conceptually simple. They seem to separate concerns, as demanded by the principles of software engineering. Unfortunately, in generating even modestly sophisticated texts the planning stage is not independent of the realization stage. In particular, since the planner isn't aware of syntax, it can't take into account opportunities and inadequacies that arise during the realization stage. For example, constraints during realization could preclude the effective expression of a concept, causing the generator to "talk itself into a corner". In this case the text plan cannot be realized, and the generator is not able to replan the utterance. Likewise, if the realization module can achieve more than one communicative goal at once — say, the word it chose encompasses two concepts — there is no way for it to tell the planner to eliminate the extraneous plan segment. This may lead to repetitive and stilted text. The context in which a sentence is realized can also affect how it is expressed. Thus the planner in pipelined systems is more or less limited to choosing the topics, establishing the sentence order, and hoping realization is successful.

Hovy argued that to avoid some of the problems of pipelined systems the planner should make only enough commitments to allow it to begin the realization process, and then pass the control and information back and forth between the modules, interleaving their execution [19]. He demonstrated the particular usefulness of this approach for generating pragmatic (non-literal) effects in text [18]. The pragmatic rules in Hovy's PAULINE were limited to coloring the text based on the interlocutors' personal characteristics, their interpersonal goals and the conversational setting, but it seems reasonable to expect that any generator which needs to achieve pragmatic effects should be capable of evaluating the effects of its speech acts. In Hovy's words, "Generators must be able to reason about *why* they say what they say," [18, p. 156].

Other examples of interleaved systems besides Hovy's PAULINE include McDonald's MUMBLE [26, 27], Kempen's IPG [11, 23], Colby's PARRY [8], Clippinger's ERMA [7], and Davey's PROTEUS [10].

Although one could, in theory, use a pipelined system to generate the same output text as an interleaved system, by taking matters of syntax and lexical selection into account, one would pay a huge computational price. Exactly the same kinds of arguments as are made in recent planning literature about the benefits of reactive planning and emergent behavior apply to text planning in natural language generation. Like a reactive planner, an interleaved generator can respond opportunistically to changes in the state of its goals and its available choices, and makes only those decisions which are directly relevant to the message it is trying to express. The interleaved architecture also permits the incremental production of output, although not all interleaved generators do so.

In our study of existing interleaved generators we began to notice that the operations used for text planning appeared similar in many ways to the operations used for surface realization, and thus wondered whether one might be able to use the same operations to accomplish both tasks. Decomposing these existing operations into primitives that distill the essence which is common to both levels yields a finer-grained set of orthogonal operations. The planning engine that processes this finer basis of factored operations unifies the modules into a single mechanism which does the work of both modules using the same set of operations.

For example, the goal ordering rules of text planning and the precedence relations of a realization grammar are both instances of "serialization". That is, they both assign a linear order, one to communicative goals and the other to syntactic constituents. Even the realization of a word involves the linear ordering of its morphemes. Thus, serialization is one operator in our unified basis.

Similarly, selection of content and selection of words are both restricted by constraints imposed during past generation. The decision of what content should be explicitly included and what should be inferred is similar to the decision whether to realize a representation in full or to use anaphora, deixis and ellipsis. The aggregation of information into paragraph and sentence groups parallels the expression of groups of content as words and phrases. Thus, "transformation" rules are another kind of primitive operation in our model.

Following this line of reasoning leads to the integrated style of generator, in which all text planning, surface realization, and other processes are accounted for within a single formalism. The generator uses a uniform representation for all the linguistic levels, and the single engine applies the same set of basic operations at each level. The levels communicate by the two-way flow of information across the boundaries between levels. The only difference between levels is isolated in the rule definitions.

To our knowledge, Appelt's KAMP system [2] comes the closest of any previously discussed generator to being integrated, at least in spirit, although it seems more like a homogeneous interleaved system in which all of the levels happen to use the same planner. For example, earlier versions of KAMP had the grammatical knowledge spread throughout the rules and operated in a semi-pipelined fashion, with critics providing limited feedback between adjacent levels. Later versions isolated surface realization into a separate component called TELEGRAM that could invoke the planner at choice points during realization to help it make a decision [1]. Nevertheless, Appelt presented the view that all generation processes, even realization, involve planning, and used a planner based on NOAH [32] to control the generation.

Certain blackboard systems, such as DIOGENES [30], while viewable as interleaved systems with more than two modules, nonetheless have much of the flavor of integrated systems. For example, the modules in DIOGENES all use the same representation language because they must communicate by posting messages to the blackboard and triggering off of the messages found on the blackboard.

We have completed a brief look at the potential benefits of integrated generation systems, and turn now to GLINDA, the integrated generator we are developing for Oz. This work is in its early stages, and both the architecture and rules are under active development. We present GLINDA here as concrete, if unfinished, example of how one might organize an integrated generator.

2. The GLINDA Integrated Natural Language Generator

In this section we survey GLINDA, presenting it as an example of an integrated generation system. We do not intend this as an in-depth description, but as a concrete illustration of the ideas discussed above. To date, work on GLINDA has focussed on implementing surface realization using operators derived from text planning systems. We believe the framework is general enough to do arbitrary text planning, given appropriately defined rules, but have not yet done so.

The Oz system is intended to let human users participate in dramatically interesting simulated microworlds that are inhabited by moderately intelligent and emotional agents. Oz is intended as the technical framework for a new interactive art form [3, 35]. GLINDA is used both for narration of interaction with an Oz world, describing locations, objects, actions, and events, and to allow computer simulated agents to communicate with each other and with human players [4, 5, 22, 25, 31].

2.1. Motivation

Oz presents a novel set of requirements for generation. Most generators are successful if they produce well-formed and lucid text. For narration, however, Oz needs to be able to tune the generation to engender subtle emotional reactions in and exert influences on the human player, and to present a variety of vivid views of the simulated world [35]. Characters must occasionally make some of the same errors and have the same idiosyncrasies in speech as human beings. The coverage of language phenomena should be broad but not necessarily deep.

Several items on our “wish list” for generation could best be characterized as multi-level linguistic phenomena with cross-level information flow and feedback. Pragmatic effects, such as those implemented in PAULINE, involve many individual local effects contributing to create a global effect. To produce drunken speech the generator needs to simultaneously make grammatical errors and apply rules that lengthen sibilants to produce slurred speech. Rhyme, alliteration, and poetry involve rewriting and editing with multi-level constraints on form. The ability to talk oneself into a syntactic corner and restart requires that the generator detect errors and backtrack across the levels to a choice point where it can begin anew.

Cross-level influences can occur even during straightforward generation. For example, the morphology of certain words depends on their syntactic category. The word “loaf+s” is

```
((type sentence) (role sentence) (mood declarative)
 ((type relation) (role matrix)
  ((type parameter) (role predicate)
   ((type word) ((type morph) (cat verb) (root die))))
  ((type parameter) (role agent)
   ((type word) (age 1)
    ((type morph) (cat noun) (root cat-1)
     (person 3) (number singular) (gender male))))))
```

Figure 2: An Example of a Group

realized as "loaves" as a noun or as "loafs" as a verb. Various kinds of agreement, including subject-verb agreement and coordinated clause tense agreement, can also be characterized as the cross-level propagation of information [22]. Producing these varied effects was the primary motivation for developing GLINDA.

We are experimenting with explicit feature propagation as a means to producing such output. For example, a pragmatic goal feature propagates throughout the generation process, causing different local effects at each level. The total effect of the many individual changes yields the desired global pragmatic effect. The integrated nature of the generator makes it easy to share knowledge between levels because the information at each level is represented using a single language and is subject to the same kinds of operators.

2.2. Representation: Groups and Features

A representation that is suited to all levels of generation must be very flexible. We use a generalized feature representation called groups. A *group* is an unordered set of items, where an *item* is a group or a feature. A *feature* is a pair consisting of a property name and a property value, `(name value)`. Features are often referred to by name. See Fig. 2 for an example of a group.

A group inside another is called a subgroup of the containing parent group. Each group must contain a `role` feature which specifies the overall function of the group within its parent. A group can have at most one role, and every subgroup of a group must have a distinct role. Every group should also contain a `type` feature to indicate the structure of the group. The type determines the roles of the group's subgroups. It also specifies the level of the group in the linguistic hierarchy (e.g., phoneme, syllable, morpheme, word, phrase, clause (relation), sentence (multi-relation) and paragraph). If no `role` feature is present, it defaults to the value of the `type` feature.

For each type of group there is a particular role which when contained in a subgroup identifies that subgroup as the *head* or *local projector* of the group. If one repeatedly selects head subgroups of a group, one eventually reaches a group that contains only features. The head feature of this group is known as the *projector* of the containing groups, and forms the core of the concept represented by the containing groups. The form `(define-group-head <type> <role>)` defines a new head.

All of the projectors used in GLINDA are arranged into an inheritance hierarchy, and all rules are suspended from constraint-tries (discrimination nets) attached to categories in the hierarchy. On occasion, the immediate parent of a projector in the hierarchy can instead be specified by including a `cat` feature in the group.

Often one may summarize a group by specifying the value of its role feature and the value of the projector feature, as if it were the feature (<role> <projector>). For example, the agent subgroup of the group shown in Fig. 2 could be abbreviated as the feature `(AGENT CAT-1)`. The full meaning of a group is the composition of the meanings of its subgroups and features, with the subgroups corresponding to elements of the representation and the features describing the relationship between the subgroups and properties of the group as a whole.

2.3. Constraints

A GLINDA rule fires when the *constraint set* on its left hand side matches a group under consideration. Matching is similar to unification, except that instead of comparing two ordered representations element by element, constraints in the pattern are tagged with roles and must match against the corresponding items of the group. If there is no constraint for a particular subgroup's role, the subgroup is ignored.

Each *constraint* in the constraint set matches against either a group or a feature. The shorthand representation for groups as a `(role projector)` pair makes them look like features, allowing groups and features to be interchangeable from the point of view of the matcher. This is the sense in which a group is just a generalized feature. Note that if a projector is specified in a constraint, it matches any projector below it in the projector hierarchy. The same is true for the role specifiers.

Constraints have the general format of `(name value)`, where name can either be the name of a property, if the constraint is to match against a feature, or the value of a role feature, if the constraint is to match against a subgroup. The value of the constraint can specify the value of the feature or the value of the projector, and may be a string, symbol, number or wildcard, or a range of values formed with disjunctions, conjunctions, and negations. Values can also be constraint sets, in which case they recursively match against the contents of the subgroup. If the value is a variable, it gets bound to either the feature value or group projector or the contents of the group, depending on the type of the variable. (The latter is useful for anaphora generation and the splicing operator described below.) The only restriction is that the binding of all variables in the constraint set must be consistent for the match to succeed.

Because GLINDA is a forward-chaining reactive system, constraints are always matched against groups, and never against expressions containing variables. In fact, each constraint is matched against a single group, so there is at most one valid set of bindings for the variables in the constraint set. This yields a match algorithm that avoids the combinatorial explosion in processing time often found in unification-based generators. Matching a constraint set against a group runs in time linear in the size of the constraint set.

2.4. Rule Formats

All rules in GLINDA have the same general format, with three distinct parts:

- CONSTRAINTS: Preconditions on the input which permit the rule to run.
- ACTIONS: Generic atomic actions, ranging from output to side-effecting the generator mechanism.
- RESULTS: Result features that are returned after running the rule to report the effects of executing the rule and impose constraints on further processing. If generation was unsuccessful, this may result in backtracking.

A rule's input specification is a conjunction of a set of constraints. The constraint sets are represented as trees, which are attached to rule-specified projectors in the class hierarchy. Because constraints are priority-ordered and occurrences in multiple rules are shared, the most significant constraints are tested first and once only. This use of structure sharing was inspired by the RETE algorithm, and yields a rule selection time which is not affected by the size of the grammar. If several rules match, the current conflict resolution strategy is to choose randomly among the most specific rules, and keep track of the other rules for possible backtracking.

2.5. Types of Operators

In examining the operations which occur at the different levels of a separated generator, we identified five distinct kinds of functions. GLINDA's grammar consists of a set of five basic types of rules corresponding to the five different kinds of operations. Each type of rule performs a single function and may be applied to any group.[2] The five basic kinds of operations are:

1. Transformation of Representation.[3]
2. Control of Information Flow.
3. Side-effecting of the Generation Mechanism.
   - Goal Ordering (serial ordered goals and parallel unordered goals).
   - Parameter Setting.
4. External Actions.
   - Text Output.
   - Generic Actions.
5. Reviewing and Repairing (Critics).

Transformation of representation rules include *mapping* rules and *splicing* rules. A mapping rule may add, delete or replace some of a group's items. For example, the phenomenon

[2] Although it is possible for a specific rule to be general enough to operate on any group, no matter what its structural (linguistic) level, often the rule constraints will restrict a rule to operating on a specific level.

[3] No relation to transformational grammar.

of do-support in English is implemented by a mapping rule which adds a modal emphatic to a group if the group contains a negative feature and an active voice feature but no modal, perfect, or progressive features:

```
(define-maprule
  :constraints ((type relation) (truth negative) (voice active)
                (modal (not *))(perfect (not *))(progressive (not *)))
  :return ((modal emphatic)))
```

Mapping rules may also be used to expand "macrofeatures" into sets of simpler features. This allows a "global" feature to have different effects in different contexts. For example, a feature describing the focus of a sentence could be operationalized by changing the voice feature of the sentence.

```
(define-maprule :constraints ((focus ?a) (agent ?a)) :return ((voice active)))
```

Mapping rules can also control the expression of general features. Because English doesn't have a morphologically realized future, the future tense feature is replaced with an `r-time not-past` feature and an `intent modal`. A splicing rule may modify some of a group's items by inserting one group within another, grouping two groups into a subgroup, or merging some groups together. Typical uses include subordination and conjunction, and the aggregation of groups into paragraphs and sentences.

Control of information flow rules are similar to mapping rules in many ways. When the generator finishes operating on a group it returns some features to provide feedback on the effects of the generation. When the generator operates on a group, it has access not only to the group but also to features passed in from the parent group and to the returns from its left sibling groups. Information flow rules specify which of these features will actually be provided to a group during its generation. For example, subject-verb agreement is accomplished in GLINDA by propagating the `number` and `person` features from the subject group to the verb group. Tenses are coordinated by propagating `r-time` features (values `past` and `not-past`) and blocking modal future features. Case-marking features are added to the flow based on the relative order of the items of a group after the group has been serialized by the goal-ordering rules. Information about recent references can be propagated to enable lexical selection rules to decide whether to use anaphora. For example, the following rules illustrate case marking, subject-verb agreement, and tense coordination:

```
(define-flow-rule noun :constraints ((role agent)(voice active))
  :pass-down ((case nominative) mood form))
(define-flow-rule noun :constraints ((case nominative))
  :pass-up (person number))
(define-flow-rule verb :constraints ((role matrix)) :pass-up (r-time))
```

The side-effecting rules come in two varieties. The first controls the goal ordering of the generator, where the items of a group are considered to be communicative goals. Usually these are *serialization* or *organization* rules, which order the items of a group according to their roles. Sometimes they are *parallelization* or *coordination* rules, which cause the group's items to be generated concurrently. Parallelization rules could be used to coordinate speech acts with other kinds of actions, such as producing sounds, gestures, winks and pauses. The second set of rules controls various parameters of the generator. For example, they can control line width, indentation, and the level of caching.

The goal-ordering process may be efficiently implemented by maintaining a tree of active serialization and parallelization rules and the corresponding local and global state of the generator.[4] To do the next operation, the generator finds its way from the root to a leaf by traversing serialization rules in a depth-first manner (expanding the head of the ordering until it completes before operating on the rest of the ordering) and the parallelization rules in a breadth-first manner (expanding each unordered goal in turn, one level at a time). Features are propagated and transformed and a rule is selected to operate on the leaf. If the rule is a primitive action, it is executed; otherwise the rule is used to continue the expansion of the tree. When the rule execution finishes or the expansion completes and the tree collapses, features are returned up the tree to update the local and global state. If we add success tests to the rules, those tests could be checked along every pass from the root to a leaf, augmenting the reactivity provided by the feature-returns.

Several serialization rules are shown below. The first realizes the agent role as the word "by" in objective case. The next two rules are default rules for the class of verbs in the projector hierarchy, and handle the organization of a clause in the active voice and the conjugation of regular verbs. These may be locally overridden by placing more specific rules lower in the hierarchy, as the next two rules demonstrate. The first generates the past tense of an irregular verb. The second illustrates how slang can be generated by substituting groups in the serialization instead of just copying subgroups. For example, if the group in Fig. 2 is generated with the `(voice active)`, `(idiom slang)` and `(time past)` features, it will be realized as the sentence "The kitten kicked the bucket." Finally, the last three rules show how progressives can be generated. The `progressive` feature is realized as a group representing the verb `begin`. The realization of the progressive feature also passes up an inflection feature, which serves as the suffix for the realization of the verbal group to the right. This inflection feature will be realized as the suffix "ing".

```
(define-organization-rule (role agent)
  :constraints ((case objective)) :order ("by"))

(define-organization-rule verb :constraints ((type relation)(voice active))
  :order (role agent r-time truth modal perfect progressive voice
               predicate object iobject))
(define-organization-rule verb :constraints ((type morph))
  :order (root suffix))

(define-organization-rule go :constraints ((type morph)(inflection past))
  :order ("went"))
(define-organization-rule die :constraints ((type relation)(idiom slang))
  :order (agent r-time truth modal perfect progressive
                ((type parameter) (role predicate)
                 ((type word) ((type morph) (cat verb) (root kick))))
                ((type parameter) (role object) (case objective)
```

[4] If the generator only needed serialization rules, it could execute them by recursively traversing the ordering, as was done in an earlier version of GLINDA [22]. The introduction of parallelization rules requires some form of concurrency, such as that provided by the iterative traversal of an active plan tree [25].

```
                  ((type word) ((type morph) (cat noun) (root bucket))
                   (number singular) (person 3) (gender neuter)(form lexical)
                   (reference definite)))))

(define-organization-rule (progressive begin)
  :order (((type word) (cat verb) (root begin))))
(define-flow-rule progressive :pass-up ((inflection pres-part)))
(define-organization-rule (inflection pres-part) :order ("ing"))
```

The operation of organization rules is similar to the scan-copy device proposed by Shattuck-Hufnagel [34]. The rules constitute a skeletal framework for the group, and the items of the group are "inserted" into the slots of the framework as the serialization operator processes the elements of the group. Organization rules can specify the order of communicative goals for textual structuring or the precedence relations in a sentence, and they can also specify the structure of words in the lexicon. In particular, a word in the lexicon is simply an organization rule with the slots already filled in with the word's morphemes.

Note that any kind of "selection" operation can be implicitly incorporated into the organization rules by the presence of constraints and the deliberate omission of a role from the ordering. Alternatively, we could define transformation rules to delete the "unselected" items from the group.

The external action rules may be used to produce output and execute actions. These rules produce text as a side-effect of the "evaluation" of the group by the generator. These rules can add a string to the cache, print the contents of the cache, or print a string without first saving it in the cache. Strings are temporarily saved in the cache before being output in order to give the reviewing and repairing rules a chance to modify them based on the next string generated.

The reviewing and repairing rules, also known as *combination* rules, are used to monitor output as it is added to the cache, and possibly modify it where the cached output and new strings are combined. The main uses of these rules have been for morphological modification of words when suffixes are added and for producing interword spacing. Two combination rules which demonstrate I-replacement and Elision are illustrated below:

```
(define-char-variable vowel "aeiou")
(define-morph-rule :name "I-Replacement"
  :left ((#\i #\y) (#\e :null)) :right ((#\i :as-self)))
(define-morph-rule :name "Elision"
  :left (((:not vowel) :as-self) (vowel :as-self) (#\e :null))
  :right ((#\e :as-self)))
```

2.6. Generation Process: Control Flow and Information Propagation

The generation process in GLINDA proceeds as follows: A particular group G is about to be processed. The generator has access to the contents of the group, the feature returns from the processing of its siblings, and some features of the group's parent, including any global features which have flowed down from the top.

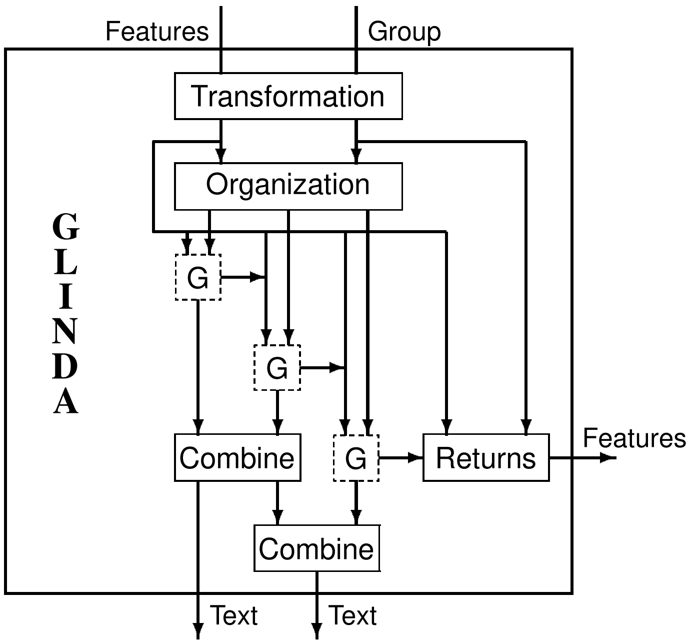

Figure 3: The GLINDA Generation Process

1. Information flow rules are selected and used to decide what features from the parent group and what features returned by the siblings will be used during the generation of the current group G.
2. Transformation rules are selected and used to modify the features and the group.
3. An organization rule is selected and used to control the recursive generation of the subgroups of the group G. The recursive generation reinvokes this whole process on the embedded subgroups.
4. External action rules are executed to produce (cached) text and other actions.
5. Combination rules are used at leaf nodes to merge the new text into the cache and incrementally output the contents of the cache.
6. Information flow rules are selected and used to decide what features will be returned for this group, based on the original features of the group and on the features returned by the recursive generation of the subgroups of the group.

Thus GLINDA uses a forward-chaining reactive style of planning which responds directly to the generator's current communication goals. This process is illustrated in Fig. 3.

2.7. An Example of Generation

Let us consider generation for a group representing a bad act by a good actor, such as George kicking the cat named Lyotard. A possible pragmatic goal feature might be to try to minimize the badness of George's actions. A transformation rule would implement this by introducing a feature that deemphasized (defocused) the actor. This feature would be propagated to the clausal level, where another transformation rule would introduce a passive voice feature because the actor is defocused and the actor is the agent of the clause. If the action can be cast in the passive voice, an organization rule which generates the subgroups of the clause in the order OBJECT ACTION AGENT would be selected. The object is realized as "Lyotard",[5] returning its person and number features. Those features are then used during the realization of the action to ensure that the verb agrees with the subject of the sentence. When realizing the action, the voice feature, which was passed down from the parent group, is realized as "was", a form of the BE verb, and returns a past-participle inflection feature. This feature is used during the generation of the predicate, and is generated as the string "ed" after the generation of the verb "kick". The generator continues with the generation of the agent. Since the agent is marked with an objective case feature, the agent role feature is realized as the preposition "by". Next the actual agent is realized as the string "George". Finally, an end-of-sentence feature is realized as a period. The full sentence generated is then "Lyotard was kicked by George."

The generation of the voice feature before the verb, the verb root before the inflection feature, and the agent role before the agent were all accomplished by organization rules. If the verb were irregular, instead of using rules attached to the `verb` class in the projector hierarchy, it would use rules attached closer to the word, either executing different inflection rules as in the case of throw/threw/thrown and hit/hit, or using rote retrieval as in the case of go/went/gone. The same mechanism of rule retrieval accounts for both regular and irregular verb inflections. Although we've glossed over a lot of details, this example should give a bit of the flavor of the generation process in GLINDA.

2.8. Performance

Our initial implementation of GLINDA yields adequate performance for our needs. The generator itself is small, approximately 10 pages of Lisp code. The current grammar contains about 600 rules, expressed as roughly 20 pages of Lisp code. In this configuration, running in CMU Common Lisp on an IBM RT-APC, a 3-MIPS machine with a good Lisp implementation, GLINDA generates 2 clauses per second (about 500 words per minute). The speed did not decrease as the rule base grew.

3. Psycholinguistic Plausibility of Integrated Systems

This section very briefly presents evidence which suggests the psycholinguistic plausibility of integrated generation systems as models of human language production behavior. The evidence — from studies of speech errors — assumes that the human language production system is a mechanism which exhibits characteristic breakdown. When the system fails, it does so in ways that show how it is put together.

Inspection of speech-error corpii such as the UCLA corpus [13, 14] and the MIT-CU corpus [15, 16, 34] readily demonstrates that speech errors are highly regular phenomena. Each constraint on the structure of spontaneous speech errors has implications for models of human

[5] A different pragmatic rule might decide to emphasize the cat's tender age in order to cause sympathy, and hence generate "The kitten" instead.

language processing. In particular, a parsimonious explanation for this data may be that the array of errors is caused by the failure of a single mechanism.

For example, speech errors involve either movement of fragments, such as shifts and exchanges, or changes in fragments, such as additions, omissions, and substitutions. Shattuck-Hufnagel observes that errors of all five basic types occur for every level of representation from phoneme through sentence constituent [34]. While one could claim that the same flaws appear in each level's independent processing module, it is simpler to suppose that the similarity of errors is caused by the flaws in a single engine which processes every level of representation.

The existence of anticipatory errors, often with several words intervening, implies that the speaker's internal representations span more than just the next word of the utterance [24]. The fragments which change, move, or are omitted in speech errors correspond to linguistically motivated units such as phonetic features, single consonants and vowels, syllables, words, phrases and clauses [13].[6] Together this suggests that language production involves planning, and that this planning employs the same units as are proposed by linguists and grammarians. Moreover, the exchanged elements must be similar and such exchanges occur within levels but not across levels, suggesting that planning operates at distinct levels of processing [34]. Because NLG researchers often cite this as evidence for distinct modules operating at each level, we will note that this evidence is consistent with a single-mechanism model in which the mechanism operates only on one kind of data at a time.

Other kinds of speech errors, such as spoonerisms, intentional distortions and tongue twisters, obey the same constraints as spontaneous speech errors. Tongue twisters, in particular, exhibit these constraints both in their inherent structure and in the errors produced when people say them. Most tongue twisters are constructed by mixing regular alternating stress patterns with irregular patterns in a way that overloads the human language production system [33]. When people stress the language production system by saying tongue twisters fast, they often regularize the patterns. Although the patterns often are phonetic, phonemic and syllabic, they can occasionally involve words and phrases,[7] suggesting again that the problems at each level are caused by the failure of a single mechanism.

4. Conclusions

The traditional view of natural language generation regards language processing as if it were separated into distinct modules, with each module responsible for the processing of the rules and representations corresponding to one or more linguistic levels. We have suggested that the modularization of the generation process does not require distinct kinds of rules and representations at each level, and indeed that there are advantages to using a single framework across all levels.

We have also described GLINDA, an integrated generator which attempts to address a wide range of natural language problems under this paradigm. GLINDA uses a uniform com-

[6] In some sense this generalizes the notion of complementary distribution in linguistics.

[7] For example, "A lump of red lead, a red lead lump."; "Knee deep, deep knee."; "A fine field of wheat, a field of fine wheat."; and the famous "Peter Piper picked a peck of pickled peppers; how many pickled peppers did Peter Piper pick?".

putational framework with explicit feature propagation and cross-level feedback to achieve multi-level textual effects. We hope to extend GLINDA to produce multi-modal output as well. While the work reported here is preliminary, we see some signs of the approach yielding a simple, elegant, and efficient generator, which is especially suited to producing pragmatic shadings in its output.

5. Acknowledgments

We'd like to thank Eduard H. Hovy for providing the inspiration behind the design of GLINDA and for commenting on a draft of this paper. Sergei Nirenburg and Bryan Loyall also provided useful comments on the draft. We'd also like to thank Robert C. Berwick for discussions of the separation of descriptions of rules and representations from the processing of the same.

References

[1] D. E. Appelt. TELEGRAM: A grammar formalism for language planning. In *Proceedings of the 8th IJCAI*, pages 595–599, Karlsruhe, West Germany, August 1983.

[2] D. E. Appelt. *Planning English Sentences*. Cambridge University Press, Cambridge, UK, 1985. Based on Stanford PhD thesis, SRI Tech Note 259, March 1982.

[3] Joseph Bates. Virtual reality, art, and entertainment. *Presence: The Journal of Teleoperators and Virtual Environments*, 1, 1992. MIT Press, Cambridge, MA. This is a revised version of *Deep Structure for Virtual Reality*, CMU-CS-91-133, School of Computer Science, Carnegie Mellon University, May 1991.

[4] Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. Broad agents. In *Proceedings of AAAI Spring Symposium on Integrated Intelligent Architectures*, Stanford, CA, March 1991. Available as *SIGART Bulletin* 2(4):38-40, August 1991.

[5] Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. Integrating Reactivity, Goals, and Emotion in a Broad Agent. Submitted to AAAI-92, San Jose, CA, January 1992.

[6] Kathryn Bock. Exploring levels of processing in sentence production. In Gerard Kempen, editor, *Natural Language Generation: New Results in AI, Psychology, and Linguistics*. Martinus Nijhoff Publ., Boston, 1987.

[7] J. H. Clippinger. *A Discourse Speaking Program as a Preliminary Theory of Discourse Behavior and a Limited Theory of Psychoanalytic Discourse*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1974.

[8] K. M. Colby. *Artificial Paranoia: A Computer Simulation of Paranoid Processes*. Pergamon Press, Oxford, 1975.

[9] Joseph H. Danks. Producing ideas and sentences. In Sheldon Rosenberg, editor, *Sentence Production: Developments in Research and Theory*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

[10] A. Davey. *Discourse Production*. Edinburgh University Press, Edinburgh, 1979.

[11] Koenraad De Smedt and Gerard Kempen. Incremental sentence production, self-correction and coordination. In Gerard Kempen, editor, *Natural Language Generation: New Results in AI, Psychology, and Ling*. Martinus Nijhoff Publ., Boston, 1987.

[12] G. S. Dell and P. A. Reich. Stages in sentence production: An analysis of speech-error data. *Journal of Verbal Learning and Verbal Behavior*, 20:611–629, 1981.

[13] Victoria Fromkin. The non-anomalous nature of anomalous utterances. *Language*, 47:27–52, 1971.

[14] Victoria Fromkin. *Speech Errors as Linguistic Evidence*. The Hague, Mouton, 1973.

[15] Merrill F. Garret. The analysis of sentence production. In G. H. Bower, editor, *The psychology of learning and motivation: Advances in research and theory*, volume 9, pages 133–177. Academic Press, New York, 1975.

[16] Merrill F. Garret. Levels of processing in sentence production. In B. Butterworth, editor, *Language Production*, volume 1. Academic Press, New York, 1980.

[17] Eduard Hovy. Integrating text planning and production in generation. In *Proceedings of the 9th IJCAI*, Los Angeles, CA, 1985.

[18] Eduard Hovy. *Generating Natural Language under Pragmatic Constraints*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1988.

[19] Eduard Hovy. Two types of planning in language generation. In *Proceedings of the 26th Meeting of the ACL*, Buffalo, New York, April 1988.

[20] P. S. Jacobs. *A Knowledge-Based Approach to Language Production*. PhD thesis, University of California at Berkeley, Berkeley, CA, 1985.

[21] P. S. Jacobs. PHRED: A generator for natural language interfaces. *Computational Linguistics*, 11(4):219–242, October-December 1985.

[22] Mark Kantrowitz. GLINDA: Natural language text generation in the Oz interactive fiction project. Technical Report CMU-CS-90-158, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[23] Gerard Kempen and Edward Hoenkamp. An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11:201–258, 1987.

[24] Karl S. Lashley. The problem of serial order in behavior. In Lloyd A. Jeffress, editor, *Cerebral Mechanisms in Behavior*, pages 112–146. John Wiley, New York, 1951.

[25] A. Bryan Loyall and Joseph Bates. Hap: A reactive, adaptive architecture for agents. Technical Report CMU-CS-91-147, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1991.

[26] D. D. McDonald. *Natural Language Production as a Process of Decision Making under Constraint*. PhD thesis, MIT, Cambridge, MA, 1980.

[27] D. D. McDonald and J. D. Pustejovsky. Description-directed natural language generation. In *Proceedings of the 9th IJCAI*, pages 799–805, Los Angeles, CA, 1985.

[28] Kathleen R. McKeown. Discourse strategies for generating natural-language text. *Artificial Intelligence*, 27:1–42, 1985.

[29] Kathleen R. McKeown. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Cambridge University Press, Cambridge, 1985. Based on PhD thesis (Univ. of Penn., May 1982), Tech Report MS-CIS-82-5.

[30] Sergei Nirenburg, Victor Lesser, and Eric Nyberg. Controlling a language generation planner. In *Proceedings of the 11th IJCAI*, pages 1524–1530, Detroit, MI, August 1989.

[31] W. Scott Reilly and Joseph Bates. Building Emotional Agents. Submitted to AAAI-92, San Jose, CA, January 1992.

[32] E. D. Sacerdoti. *A Structure for Plans and Behavior.* North Holland/American Elsevier, 1977. Based on PhD thesis, SRI Tech Note 109, August 1975.

[33] L. Schourup. Unique new york unique new york unique new york. In *Papers from the 9th Regional Meeting, Chicago Linguistic Society*, pages 587–596, 1973.

[34] Stefanie Shattuck-Hufnagel. Speech errors as evidence for a serial-ordering mechanism in sentence production. In W. Cooper and E. Walker, editors, *Sentence Processing: Psycholinguistic Studies Presented to Merrill Garrett*, pages 295–342. Lawrence Erlbaum Associates, Hillsdale, NJ, 1979.

[35] Sean Smith and Joseph Bates. Toward a theory of narrative for interactive fiction. Technical Report CMU-CS-89-121, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[36] J. P. Stemberger. An interactive activation model of language production. In Andrew W. Ellis, editor, *Progress in the Psychology of Language*, volume 1. Lawrence Erlbaum and Associates, 1985.