SURGE: a Comprehensive Plug-in Syntactic Realization Component for Text Generation

Michael Elhadad
http://www.cs.bgu.ac.il/~elhadad
elhadad@cs.bgu.ac.il
Mathematics and Computer Science Dept.
Ben Gurion University in the Negev
Beer Sheva, 84105 Israel
Fax: +972-7 472-909

Jacques Robin
http://www.di.ufpe.br/~jr
jr@di.ufpe.br
Departamento de Informática
Universidade Federal de Pernambuco
Caixa Postal 7851, Cidade Universitária
Recife, PE 50740-540 Brazil
Fax: +55-81 271-8438)

SURGE is a syntactic realization front-end for natural language generation systems. It is aimed to be used as a reusable tool encapsulating a rich knowledge of English syntax. We explain how the wide coverage necessary to reach the goal of reusability has been reached through (1) a pragmatic approach of integrating several linguistic theories as required, and (2) a level of abstraction for the input description that does not overcommit the other components of the generator to specific ontological perspectives.

The paper describes how a generator can interact with SURGE by preparing input specifications. The input description language is presented and motivated and different methods to construct the specifications are contrasted.

Finally, the coverage of SURGE is discussed and techniques to extend it are presented, including a short description of the development tools provided to the grammar writer.

The development and maintenance of the SURGE system over the past decade have demonstrated that (1) syntactic realization becomes difficult to manage as the grammar grows, (2) wide coverage requires attention to categories and constructs often ignored in traditional grammatical theories (e.g., dates, addresses, person names), and (3) integration into many different systems has strongly motivated a relatively low level of abstraction as an interface between domain-specific system components and a reusable syntactic realization component.

1. Introduction

While generators vary widely in the function they fulfil, in the source data they use as a starting point and in their architecture, they all end up producing text which follows and exploits the syntax of the target language. This indicates that a syntactic realization component can be developed as a reusable resource, which can be plugged in into a wide range of systems. We report in this paper on the development and evolution of such a component, the SURGE system. SURGE has been widely distributed in the generation community and has been embedded into several complete systems. The goals of reusability and wide coverage have led to a large system and many of the issues faced during the development of the system are issues common to the development of large software systems. From the linguistic side, they have also led to the definition of an input specification language of a relatively low level of abstraction and to a pragmatic approach to grammar development that integrates several distinct theories into one computational engine. Indeed, while overall SURGE is organized around the lines of a systemic functional grammar, we have found it efficient to borrow from other linguistic schools (HPSG (Pollard and Sag, 1994)) and descriptive works (Quirk et al., 1985) to implement different aspects of the grammar (for example, control, syntactic pronominalization and binding). We will describe the internal organization of SURGE in Section 5. However, the main purpose of the paper is to explain the process of embedding SURGE in complete generation systems and to demonstrate the services SURGE provides to the generator.

The paper is organized as follows: Section 2 illustrates the input specification language to SURGE informally through a simple example. In Section 3, we explain how the objective of reusability has led to design guidelines for the development of SURGE. Section 4 describes the interface between a client program and SURGE by presenting the input specification language and three different methods to construct specifications. Section 5 presents the internal organization of the grammar and its linguistic structure. Finally, Section 6 presents closing observations on the development environment for large grammar maintenance and compares SURGE with similar syntactic realization components.

2. A simple example

SURGE is used to map input specifications to English sentences. We begin our presentation of SURGE with an informal example of such an input specification. In general, such input specifications are constructed by application programs (we explain how in Section 4). In this section, however, we will illustrate the process of *manually* specifying an input for a given sentence.

SURGE accepts as input thematic trees with all content words[1] specified. These trees are represented as feature structures encoded in the FUF formalism, our implementation of the functional unification formalism (Elhadad, 1993a). Such feature structures are called *functional descriptions* or FDs. FDs can be thought of as sets of attribute-value pairs, which can be recursively embedded. Such structures can be represented either as graphs or as feature structure matrices, as shown in Fig.1 and Fig.2. Each pair in an FD is called a feature. Note that the order of appearance of features within FDs is of no significance.

Figure 1 is an input specification for the sentence: "*Michael Jordan scored 36 points.*" At the toplevel, the FD includes the syntactic category of the sentence, in the feature `(cat clause)`. Because it is a clause, and following the systemic terminology, the FD is then organized around the process feature. The process contains the lexical entry of the main verb of the clause (*to score*). It also includes lexical information about this verb: "*to score*" is classified as a `material` and `creative` verb. This classification determines which roles can participate in the event denoted by this process. These participants eventually surface as subject and object in the clause. The `process` feature also includes the tense of the denoted event (past). A different process type (*e.g.*, possessive) would require the use of different participants (possessor and possessed).

Another general characteristic of the input FD, is that it is organized around linguistic constituents: the clause is analyzed as the following constituent structure: *[Michael Jordan] [scored] [36 points]*. Each constituent is labelled *functionally* — *i.e*, the role of the constituent within the larger context is explicitly specified (as process, agent or created). (This labelling is the main explanation of the term "functional description".)

1 *i.e.,* nouns, adjectives and semantically full verbs, as opposed to closed-class or function words such as articles, pronouns, conjunctions and auxiliary or support verbs.

$$\begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \begin{bmatrix} \text{type} & \text{material} \\ \text{effect-type} & \text{creative} \\ \text{lex} & \text{“score”} \\ \text{tense} & \text{past} \end{bmatrix} \\ \text{participants} & \begin{bmatrix} \text{agent} & \begin{bmatrix} \text{cat} & \text{proper} \\ \text{head} & \begin{bmatrix} \text{cat} & \text{person-name} \\ \text{first-name} & \begin{bmatrix} \text{lex} & \text{“Michael”} \end{bmatrix} \\ \text{last-name} & \begin{bmatrix} \text{lex} & \text{“Jordan”} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{created} & \begin{bmatrix} \text{cat} & \text{np} \\ \text{cardinal} & \begin{bmatrix} \text{value} & 36 \end{bmatrix} \\ \text{definite} & \text{no} \\ \text{head} & \begin{bmatrix} \text{lex} & \text{“point”} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 1
Input specification I_1 for the sentence S_1 *“Michael Jordan scored 36 points.”*

Each constituent is in turn characterized by a syntactic category — `(cat np)` for the two participants — and is organized around a main feature, often called the “head,” which determines which features are appropriate at each level. Some constituents are labelled with purely syntactic categories (like NP), others with more semantic categories (like person-name).

The general aspiration when designing an input specification language for a syntactic realization component is to make it as semantic as possible, so that the gap between the knowledge representation language of the underlying system and the syntactic representation be as reduced as possible. From this perspective, a category like person-name is quite desirable, since it is most likely that all knowledge sources used in generators have a semantic characterization of persons and a representation of person names in their ontology. A feature like (cat proper), on the other hand, is less desirable, because it requires the system preparing the FD to know more about syntax. The first objective when developing an input specification language is thus to raise the abstraction level. On the other hand, if the syntactic realization component is to be portable among a wide range of domains and systems with different ontologies, it must avoid to become over-committed to a specific ontology, which could be different from the one used in different domains. The second objective which constrains the design of the input specification language is thus to preserve portability, and it somewhat conflicts with the first one. We discuss and motivate in Sect.3 and Sect.4 the specific solution to this trade-off implemented in SURGE.

This first example illustrates the following type of information encoded in SURGE's input: the constituent structure (process and participants), the syntactic category of each constituent (clause or np) and the content words for the head of each constituent (*“score”*, *“point”*).

Many default features are not specified in the input and are filled in by the grammar as generation proceeds. For example, the category of the head of the `created` participant is a common noun, but this is chosen as a default by SURGE and it does not need to be specified. Similarly, the number of `created` is plural, but this is inferred by SURGE because the `cardinal` feature is specified and its value is not 1.

The generation of a sentence from such an input FD proceeds in two main stages:

1. The input FD is unified with the grammar, which is itself encoded as a large feature structure, including disjunctions. As a result, the input FD is enriched by features of the grammar and the relevant syntactic knowlegde is

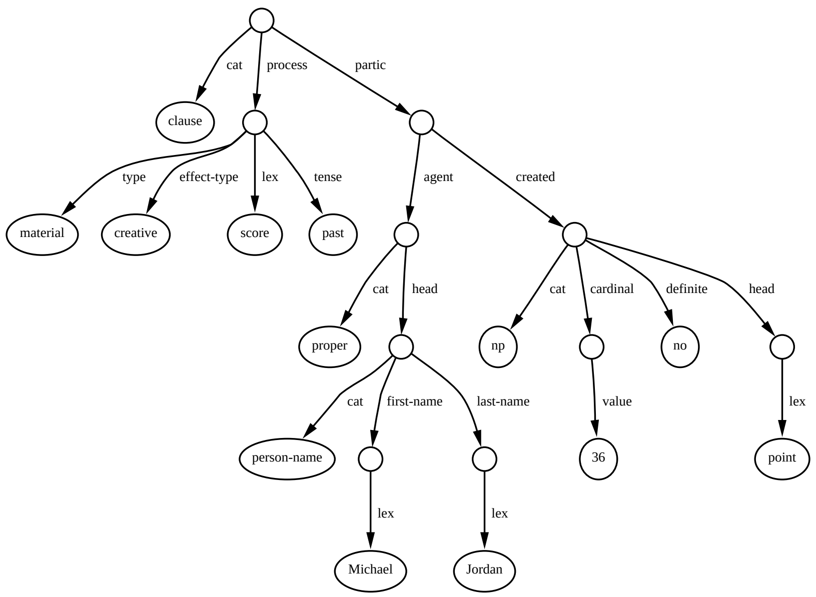

Figure 2
Graph representation of the I_1 matrix in Fig. 1

incorporated into the FD. The resulting FD encodes both the input semantic specification and the syntactic features selected by the grammar. It is a complete and consistent set of default values for closed-class words, syntactic features and constituent ordering constraints.

2.Once unification completes, the FUF interpreter turns the complete syntactic description into a linear string of tokens annotated with morphological and punctuation features (*e.g.*, verbs are annotated with person, tense, number), and the morphologizer turns this string into a complete sentence. For example, from the root words "score" and "point" this second stage derives the tokens "scored" and "points" and adds a period at the end of the sentence.

Once an input is specified for a sentence, it can be incrementally refined to define a more complex sentence. For example, Figure 3 illustrates how the first input of Fig. 1 can be enriched to describe the more complex sentence: *"Michael Jordan scored two-thirds of his 36 points with 3 point shots."* (The new features are shown in bold face.)

First, a whole new constituent is added as the feature **`pred-modif|instrument`**, encoding the fragment "*with 3 point shots*." This constituent is attached to the clause, but is a modifier less central to its meaning — it is called in SURGE a predicate modifier, following Quirk's terminology (Quirk et al., 1985, pp.504-505). As usual, the syntactic category of the instrument role is specified in the input, with the feature `(cat pp)` for a prepositional phrase. The preposition itself, however, is not specified. It is derived by SURGE from the functional labelling of the modifier as an instrument. This labelling is also used to determine where to position the constituent within the clause, with respect to participants and to other modifiers if they were present. It would also be used to determine how to formulate a question (*"how"* asks about the instrument, as shown below).

$$
\begin{bmatrix}
\text{cat} & \text{clause} \\
\text{process} & \begin{bmatrix} \text{type} & \text{material} \\ \text{effect-type} & \text{creative} \\ \text{lex} & \text{“score”} \\ \text{tense} & \text{past} \end{bmatrix} \\
\text{partic} & \begin{bmatrix}
\text{agent} & \begin{bmatrix} \text{cat} & \text{proper} \\ \text{head} & \begin{bmatrix} \text{cat} & \text{person-name} \\ \text{first-name} & \begin{bmatrix} \text{lex} & \text{“Michael”} \end{bmatrix} \\ \text{last-name} & \begin{bmatrix} \text{lex} & \text{“Jordan”} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\
\text{created} & \begin{bmatrix} \text{cat} & \text{np} \\ \textbf{fraction} & \begin{bmatrix} \textbf{num} & \mathbf{2} \\ \textbf{den} & \mathbf{3} \end{bmatrix} \\ \textbf{possessor} & \begin{bmatrix} \textbf{cat} & \textbf{personal-pronoun} \\ \textbf{gender} & \textbf{masculine} \end{bmatrix} \\ \text{cardinal} & \begin{bmatrix} \text{value} & 36 \end{bmatrix} \\ \text{head} & \begin{bmatrix} \text{lex} & \text{“point”} \end{bmatrix} \end{bmatrix}
\end{bmatrix} \\
\textbf{pred-modif} & \begin{bmatrix} \textbf{instrument} & \begin{bmatrix} \textbf{cat} & \textbf{pp} \\ \textbf{np} & \begin{bmatrix} \textbf{classifier} & \begin{bmatrix} \textbf{cat} & \textbf{measure} \\ \textbf{quantity} & \begin{bmatrix} \textbf{value} & \mathbf{3} \\ \textbf{digit} & \textbf{yes} \end{bmatrix} \\ \textbf{unit} & \begin{bmatrix} \textbf{lex} & \textbf{“point”} \end{bmatrix} \end{bmatrix} \\ \textbf{number} & \textbf{plural} \\ \textbf{definite} & \textbf{no} \\ \textbf{head} & \begin{bmatrix} \textbf{lex} & \textbf{“shot”} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{bmatrix}
$$

Figure 3
Input specification I_2 for the sentence S_2 *“Michael Jordan scored two-thirds of his 36 points with 3 point shots”*

The second enrichment to the base sentence is concentrated within one NP constituent: the `created` role. The features fraction and possessor are added and the NP is changed from: *36 points* to *two-thirds of his 36 points.* SURGE introduces the predeterminer fraction with the function word *“of”* as required, and generates the personal possessive pronoun *“his”* out of the features `category` and `gender`.

In this first modification, the possessor is simply specified as a masculine personal-pronoun. There is no indication that the possessor of the points happens to co-refer with the agent of the clause (Michael Jordan). A more abstract input specification for the same sentence is shown in Fig. 4 (with the details of the instrument omitted). In this case, the `index` feature is added to the `agent` role and a semantic concept identifying the denoted agent (tagged `concept mj1`) is specified. The possessor of the `created` role is then only specified in terms of its `index`, which is shared with the index of the agent (agent and possessor are co-indexed). The number tag [1] in the input indicates structure sharing (reentrancy) within the FD, as is customary in unification formalisms. Processing this input specification requires much more effort from SURGE: it determines that the two constituents (`agent` and `created|possessor`) are coindexed and that they fit in a configuration which triggers obligatory pronominalization (as indicated by the theory of binding discussed in (Pollard and Sag, 1994) Chapter 4) and therefore decides to pronominalize the possessor (eventually producing *“Jordan scored two-thirds of his 36 points”* instead of *“Jordan scored two-thirds of Jordan's 36 points”* which would be misleading). How this decision is implemented is discussed in Sect.5.

Interestingly, it is not clear that this higher level of abstraction in the input is always more desirable. Other components of the generator may be better placed to determine that a pronoun should be used for the possessor, using less complex processing. In this

$$\begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \boxed{\ldots \textit{cf. } I_2 \textit{ in Fig.3}} \\ \text{partic} & \begin{bmatrix} \text{agent} & \begin{bmatrix} \textbf{index} & [1]\begin{bmatrix}\textbf{concept} & \textbf{mj1}\\ \textbf{gender} & \textbf{masculine}\end{bmatrix} \\ \text{cat} & \text{proper} \\ \text{head} & \begin{bmatrix} \text{cat} & \text{person-name} \\ \text{last-name} & \begin{bmatrix}\text{lex} & \text{“Jordan”}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{created} & \begin{bmatrix} \text{cat} & \text{np} \\ \text{fraction} & \begin{bmatrix}\text{num} & 2 \\ \text{den} & 3\end{bmatrix} \\ \text{possessor} & \begin{bmatrix} \textbf{index} & [1] \\ \text{cat} & \text{proper} \\ \text{head} & \begin{bmatrix} \text{cat} & \text{person-name} \\ \text{last-name} & \begin{bmatrix}\text{lex} & \text{“Jordan”}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{cardinal} & \begin{bmatrix}\text{value} & 36\end{bmatrix} \\ \text{head} & \begin{bmatrix}\text{lex} & \text{“point”}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\ \text{pred-modif} & \boxed{\ldots \textit{cf. } I_2 \textit{ in Fig.3}}\end{bmatrix}$$

Figure 4
Alternative, more semantic input specification I_2^b for the sentence S_2

$$\begin{bmatrix} \text{cat} & \text{clause} \\ \textbf{mood} & \textbf{wh} \\ \textbf{scope} & [1] \\ \textbf{proc} & \begin{bmatrix}\textbf{type} & \textbf{mental} \\ \textbf{lex} & \textbf{“think”}\end{bmatrix} \\ \textbf{partic} & \begin{bmatrix} \textbf{processor} & \begin{bmatrix} \textbf{cat} & \textbf{personal-pronoun} \\ \textbf{animate} & \textbf{yes} \\ \textbf{person} & \textbf{second}\end{bmatrix} \\ \textbf{phenomenon} & \begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \boxed{\ldots \textit{cf. } I_2 \textit{ in Fig.3}} \\ \text{partic} & \begin{bmatrix} \text{agent} & [1]\boxed{\ldots \textit{cf. } I_2 \textit{ in Fig.3}} \\ \text{created} & \boxed{\ldots \textit{cf. } I_2 \textit{ in Fig.3}}\end{bmatrix} \\ \text{pred-modif} & \boxed{\ldots \textit{cf. } I_2 \textit{ in Fig.3}}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 5
Input specification I_3 for the sentence S_3 *"Who do you think scored two-thirds of his 36 points with 3 point shots?"*

case, SURGE supports both approaches, leaving the decision to the architect of the generator. The different options are discussed in Section 4. A similar argument is developed in (Dale, 1995) regarding the decision to use one-pronominalization.

As a final example of the services provided by SURGE, we illustrate how the input specification can be compositionally extended to describe complex sentences simply by combining the building blocks presented above. In Fig. 5, the input for *"Who do you think scored two-thirds of his 36 points with 3 point shots?"* is composed by embedding the input FD shown in Fig. 3 into a matrix FD for the clause "you think that ..." The mood of the matrix clause is set to a wh-question, and the scope of the question is set to the agent of the embedded clause.

By simply changing the scope of the question to be the instrument modifier of the embedded clause, one would get the following sentence:

How do you think that Michael Jordan scored two-thirds of his 36 points?

In summary, this example has demonstrated the main services provided by SURGE:

- It provides a compositional input specification language that interfaces between an application program and knowledge of the syntax. Input specifications include a thematic structure (process, participants and modifiers), the syntactic category of all major constituents (clause, np), and all content words.
- SURGE provides defaults for most features. As the number of possible features at each level is quite large, this is a critical point.
- It determines function words based on functional description (pronouns, prepositions, determiners).
- It selects an appropriate ordering among the constituents of the sentence.
- It propagates agreement among constituents and computes morphological inflections.
- It decides when syntactic pronominalization must be performed.

A more detailed list of the tasks fulfilled by a syntactic realization component is provided at the end of the paper. The critical design decision faced when developing such a component is the selection of an appropriate abstraction level for the input description. In the case of SURGE, this has been dictated by our concern for reusability, an issue we now turn our attention to.

3. Re-Usable Components in NLG

In the most general case, a natural language generator takes as input both (1) a representation of some semantic content and (2) a representation of some communicative goals. Both parts of this input come from the underlying application domain. The overall task of the generator is to produce a sentence or text that:

- Conveys the input semantic content.
- Satisfies the input communicative goals.
- Is linguistically correct for the natural language(s) in use.

This overall task can therefore be decomposed in four distinct sub-tasks:

- *Content Determination* ("what to say?") which consists of selecting the part of the input semantic content that is relevant to the input communicative goals.
- *Content Organization*, ("when to say what?") which consists of determining how to group, relate and order the selected content units inside linguistic units at various ranks (paragraphs, sentences, syntactic constituents).
- *Lexicalization*, ("how to say it?") which consists of choosing the main words and high level syntactic forms to express each content unit.
- *Syntactic Realization*, ("say it correctly") which consists of filling in the lexical and syntactic details and produce the final NL strings, while enforcing the linguistic constraints of the natural language in use (grammar rules, lexical syntactic constraints, lexical co-occurrence restrictions, etc.).

In the literature, content determination and content organization together have been referred to as "content planning", "strategic generation" or "deep generation" while lexicalization and syntactic realization together have been referred to as "linguistic realization", "tactical generation"or "surface generation".

The above decomposition is a conceptual one, valid for any application domain and independent of the generator's architecture. The architecture does not define what these sub-tasks are but merely how they are distributed among the generator's components.

Encapsulating Syntactic Knowledge. The question of portability in text generation is therefore the following: for which of these sub-tasks, can all the necessary knowledge be encapsulated in a plug-in component re-usable for any real scale application?

In the current state of the art in ontological knowledge representation, rhetorical theory and computational semantic lexicography, the syntactic realization sub-task is a first candidate for such full-fledged portability. This is because while texts in different domains talk about different situations and entities, follow different organizational patterns[2] and use different vocabularies, their sentences satisfy the same linguistic constraints (as long as they use the same natural language).

The constrast between the generators ADVISOR-II (Elhadad, 1995) and ANA (Kukich, 1983) illustrates the reusability of the syntactic component across different domains and applications. ADVISOR-II generates answers to students questions about what courses they should take a given semester. ANA generates daily summaries of stock market fluctuations. These two applications completely differ in terms of:

- *Communicative goal.* ANA has an implicit, fixed communicative goal: to concisely summarize the input information (half-hourly update of financial indicators). In constrast, ADVISOR-II's communicative goal is explicitly given as input: to encourage or discourage the student to take a particular set of courses. The two ADVISOR-II outputs shown in Fig. 7 were generated as answers to the same question from the same student, but with opposite communicative goals.
- *Domain ontology.* The only situations described by ANA are factual events concerning financial indicators (namely, how they go up and down over a day). In contrast, the situations described by ADVISOR-II include (1) objective relations between courses, topics and students from an academic knowledge base, as well as (2) subjective evaluations of these relations with respect to the input communicative goal.
- *Output text structure.* As shown in Fig. 6, ANA generates two paragraphs made of complex sentences in order to achieve conciseness and follow the style of written news summaries. As shown in Fig. 7, ADVISOR generates in constrast a single paragrap made of simpler sentences, as appropriate for a turn in a pedagogical dialog.
- *Vocabulary.* Almost none of the content words used in ANA's stock market summaries are also used in ADVISOR-II's academic advice and vice-versa. In addition, the few common words tend to be used to express different semantic concept in each domain. They must therefore be stored as different entries in the necessarily concept-indexed lexicons used in generation.

2 While attempts such as Rhetorical Structure Theory (Mann and Thompson, 1987) have been made to define a general set of building blocks from which to assemble patterns in different domains, no one denies that the patterns themselves are highly dependent on the communicative goals and ontology of the discourse domain.

Thursday, June 24 1982 – Wall Street's securities markets meandered upward through most of the morning, before being pushed downhill late in the day yesterday. The stock market closed out the day with a small loss and turned to mixed showing in moderate trading.
The Dow Jones average of 30 industrials declined slightly, finishing the day at 810.41, off 2.76 points. The transportation and utility indicators edged higher. Volume on the Big Board was 55,860K shares compared with 62,710K shares on Wedsnesday. Advances were ahead by about 8 to 7 at the final bell.
Figure 6
Example output of ANA

Communicative goal = `take(student,AI)`:
Output text = *"AI covers many interesting topics such as NLP, Vision and Expert Systems. And it involves a good amount of programming. So it should be interesting".*
Communicative goal = `not(take(student,AI))`:
Output text = *"AI covers logic, a very theoretical topic, and it requires many assignments. So it could be difficult".*
Figure 7
Example outputs of ADVISOR-II

Hence, working in different domains, ANA and ADVISOR accept completely different inputs and aim at producing completely different target texts. They therefore can hardly re-use the same set of rules for content determination, content organization and lexicalization. However, since they both use English they can rely on the same set of rules to produce grammatical output. For example, both generators need to perform subject-verb agreement and they both need to choose from the same small set of articles, pronouns and conjunctions available in English. In fact, the system COOK (Smadja and McKeown, 1991), which generates the same reports as ANA in a more compositional manner, uses, like ADVISOR, SURGE as a plug-in component for syntactic realization.

The FUF/SURGE *Approach to Reusability.* In a nutshell, NLG consists of gradually mapping a purely conceptual, domain-specific, input representation, into a series of intermediate representations that become more linguistic and domain-independent at each level of abstraction. Having identified syntactic realization as the first generation sub-task to encapsulate in a fully domain-independent plug-in component, three key questions then arise:

- *Input.* What should be the precise scope of the generation sub-tasks delegated to this plug-in Syntactic Realization Component? In other words, what is the right level of abstraction for its input representation? On the one hand, it must be abstract enough to maximize the services the syntactic realization component provide to the client program (*i.e.,* the other generation components). On the other hand, it must be linguistic enough to remain invariant when moving across application domains. What is the best trade-off between these two conflicting goals?

- *Coverage.* What is the best development methodology to achieve the very wide syntactic coverage necessary to make extensions for each new target output sub-language only marginal in size? While the grammar rules of any natural language are much fewer than its word senses or expressable concepts[3] they are still very many. The quality of a plug-in syntactic realization component will

3 Which is precisely what makes syntactic realization so much more easily portable than content planning and lexicalization.

thus very much depend on the comprehensiveness with which it encodes the syntax of a natural language.

- *Practicality.* What are the desirable properties for a syntactic realization component to constitute a genuinely practical tool for developing NLG applications?

The FUF/SURGE approach to re-usability is based on four main principles:

- *Isolation of domain-independent knowledge*: SURGE attempts to encapsulate only truly domain-independent knowledge though as much of it as possible.

- *Linguistic theory integration*: SURGE combines treatments of syntactic phenomenon taken from several different linguistic theories, as well as derived from theory-neutral descriptive linguistics. SURGE integrates, within the computational framework of functional unification, linguistic analyses from a variety of theories, from both the systemic and lexicalist traditions. Specifically, we used the general thematic structure classes proposed by systemic linguists (Fawcett, 1987) and (Halliday, 1994) to define the input of the majority of sentences. However, for small classes of verbs whose semantic and argument structure do not fit this classification and which exhibit complex syntactic behavior, such as subject or object control verbs, we used the analysis proposed by lexicalist linguists (Pollard and Sag, 1994) whose work also focuses on the related phenomema of control and binding.

  In spite of this theory integration effort, many sentences that we encountered in the sub-language of practical generation applications contained constructs for which no analysis grounded within a linguistic theory were available. For such cases, we drew largely on theory-neutral descriptive syntax references such (Quirk et al., 1985). This is the case, for example, for a significant numbers of adverbial modifiers that SURGE covers for the generation of complex sentences. In some cases though, even theory-neutral references did not cover the constructs we found in the target text corpus of a particular application and we had to design our own analysis from the regularities we observed in the corpus. This was the case, for example for some of the complex NPs we found in sports newswire reports, where pre-modifiers appeared *before* numerical constituents (*e.g., "a career high 41 points"*).

- *Application-driven incremental development*: SURGE evolved from a modest bootstrapping version, to its current comprehensive version, through a series of gradual extensions, all triggered by the need to cover new syntactic constructs encountered in the sub-languages of practical generation applications.

- *Flexible, scalable computational formalism*: SURGE was implemented in the Functional Unification Formalism FUF, for its high degree of expressive flexibility and its ability to smoothly scaling-up from small prototype to very large system.

Application-driven development. To date, SURGE has been used as a syntactic realization server by at least a dozen complete generation systems. Two of these applications served as concrete anchors for successive large-scale extensions of SURGE's coverage starting from the bootstrapping version implementing the English grammar outlined in (Winograd, 1983).

ADVISOR-II (Elhadad, 1993b), which generates query responses in an interactive student advising system. It focuses on presenting the same underlying content as argument towards different conclusions and on expressing the same piece of content across linguistic ranks (from complex clause down to the determiner sequence). It prompted the development of SURGE's sophisticated determiner sequence and connective sub-grammars, including their rich set of controlling input features. It also prompted the re-thinking of the transitivity system responsible in the clause sub-grammar for mapping thematic roles onto syntactic ones. The first extension to this system was the integration of Fawcett's unified analysis of three role thematic structures as causal superpositions of two two-role structures sharing a common role (Fawcett, 1987). This system elegantly and concisely captures many of the causative and ergative alternations discussed in (Levin, 1993). However, even such better principled and extended system proved too narrow to handle many verbal argument structures encountered in student-advising dialogs (and other types of texts such as biological definitions (Lester, 1994a)). To this effect, the meta-class of "lexical roles" was introduced. These differ from thematic roles in that their mapping to oblique roles[4] must be specified in the input instead of being computed by SURGE. The idea is to delegate such mapping to SURGE's client program for clauses whose thematic structures are either (1) idiosyncratic or (2) not yet neatly integrated to SURGE's thematic ontology.

ADVISOR-II composes a SURGE input by using FUF to unify an input semantic network (enriched with argumentative constraints) with a declarative, word-based lexicon. It relies on a complex, input driven control strategy involving goal freezing and dependency-directed backtracking which is described in (Elhadad, McKeown, and Robin, 1997). The input preparation strategy is briefly described in Sect.??.

STREAK (Robin, 1994), which generates newswire style summaries of basketball games from game statistics and related historical data. It focuses on usage of concise linguistic forms, generation of very complex sentences, high paraphrasing power and empirical evaluation of architecture and knowledge structures based on corpus analysis. It prompted a dramatic extension to the adverbial system of the clause sub-grammar as well as to the variety and complexity of sub-constituents inside the NP. While rooted in the particular observations made in STREAK's target text corpus of basketball game summaries, those extensions revealed very useful for all subsequent summarization applications of SURGE in other domains (such a PLANDOC, FLOWDOC and ZEDDOC described in Appendix). This is because packing many facts in complex linguistic constituents is a pervasive strategy to achieve conciseness and fluency.

It incrementally composes the SURGE input using a revision-based control strategy. It first composes a draft of this input by using FUF to unify a semantic network of obligatory facts with a phrase planner and then a lexicon. This draft is then incrementally revised (at times non-monotonically) to opportunistically incorporate as many optional or background fact that can fit under corpus-observed space and linguistic complexity limits (Robin and McKeown, 1996).

Eight other applications using SURGE are briefly described in Appendix, demonstrating the wide variety of domains and architectures within which it has been embedded.

Flexible, scalable computational formalism. SURGE is implemented in the special-purpose programming language FUF (Elhadad, 1993a) and it is distributed as a package with a FUF interpreter. FUF is an extension of the original functional unification formalism put forward by Kay (Kay, 1979). It is based on two powerful concepts: encoding knowledge in

4 Oblique roles are described in Section 5.2.

recursive sets of attribute value pairs called *Functional Descriptions* (FD) and uniformly manipulating these FDs through the operation of unification.

Both the input and the output of a FUF program are FDs, while the program itself is a meta-FD called a *Functional Grammar* (FG). An FG is an FD with disjunctions and control annotations. Control annotations are used in FUF for two distinct purposes:

- To control recursion on linguistic constituents: the input FD is fleshed out in top-down fashion by re-unifying each of its sub-constituent with the FG.
- To reduce backtracking when processing disjunctions.

The key extensions of FUF over Kay's original formalism are its wide range of control annotations (Elhadad and Robin, 1992) and its special constructs to define subsumption among atomic values (Elhadad, 1990). The main advantages of FUF over PROLOG for natural language generation are:

- Partial knowledge assumption (a consequence of relying neither on predicate arity nor on attribute order when encoding information in FDs).
- Default control flow adapted to generation (top-down breadth-first recursion on linguistic constituents) and possibility to fine-tune it (indexing on grammatical features and dependency-directed backtracking and goal freezing on uninstantiated features).
- Built-in linearizer and morphology component.

Besides its use in SURGE, FUF has been used for other aspects of the generation process (content planning, lexical choice, media coordination for multimodal generation) and has proven a robust and scalable formalism. Some specific properties of the formalism make it particularly appropriate for implementing syntactic realization:

- Regular input: because all information is encoded uniformly as FDs, input specifications can be easily composed as building blocks. Because other components of generators (lexical choosers and even content planners) can also be implemented in FUF, a uniform formalism is also beneficial.
- Bidirectional: information can be arbitrarily exchanged between the input and the grammar. For example, in SURGE, subcategorization information for a verb can be either specified in the input (for lexical processes), or extracted from the grammar (for process types defined in the transitivity system).
- Supports partially canned input: arbitrary constituents can be specified as strings instead of as fully detailed inputs. This is useful when the underlying program manipulates macro-concepts as ready-made expressions.
- Flexible order of decision making: interacting constraints can force decisions to be made in different orders depending on the context. We show in Sect.5 for example how the voice of a clause (active vs. passive) can be determined at different stages of the sentence generation process because of co-reference constraints.

4. The Input Specification Language

Generators exploit SURGE's services by preparing inputs. Syntactically, inputs are feature structures. The SURGE input specification language determines the set of attributes and their possible values and structure. This section presents the main elements of this language and presents three different strategies generators can use to build input specifications.

4.1 Structure of the Input

Overall, input specifications for clauses are thematic trees with all content words specified and the syntactic category of the main constituents specified. SURGE's main categories are: clause, nominal group (or NP), prepositional phrase (PP) and adjectival phrase. The first step in specifying an input is to provide its category, in one of the following forms: `((cat clause) ...)`, `((cat np) ...)`, `((cat pp) ...)` or `((cat ap) ...)`.

The structure of the input depends then on the category feature. We only detail in this article the main features of the clause to illustrate the process of building input specifications.

There are three types of features involved when preparing a clause specification, corresponding to Halliday's tri-partite organization in inter-personal, ideational and textual meta-functions. According to the systemist perspective adopted in SURGE, a clause can be described simultaneously as a message (textual), as an exchange between a listener and a hearer (inter-personal) and as representation (ideational) (Halliday, 1994, p.34).

On the ideational dimension, a clause is a configuration of roles around a central process realized in language. Process is used here in the systemic sense and has a very general interpretation - independent of the contrast event/process. Processes include relations, states, events and actions. For example, the possessive process type is characterized by the combination of two participants: possessor and possessed. The syntax of the clause determines how a given process type is to be realized by a linguistic structure by specifying in which order constituents appear, the category of each constituent (NP, embedded clause, PP or adverbial), and imposing certain realization features on governed constituents (prepositions of PPs, mood of embedded clauses).

The process feature includes the lexical entry for the main verb of the clause and its type. The process type determines which set of participants are expected in the input. In addition, the process can include information on the tense and polarity of the clause. Tenses can be specified either by name (selecting one of the 36 tenses listed in (Halliday, 1994, pp.198–207)) or semantically in terms of Allen's temporal relations (Allen, 1983) (*e.g.,* to describe a past event, the generator provides the feature `(tpattern (:et :before :st))`, specifying that the event time (et) precedes the speech time (st)).

At this stage, a clause input contains the following features (any feature can be omitted in which case SURGE fills in an appropriate default):

$$\begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \begin{bmatrix} \text{type} & < \text{type} > \\ \text{lex} & < \text{verb} > \\ \text{tense} & < \text{tense} > \\ \text{polarity} & < \text{polarity} > \end{bmatrix} \\ \ldots & \end{bmatrix}$$

The roles attached to the process are determined by its type, according to the rules of the transitivity system of the grammar, which are summarized in Table 1 and Table 2 in Appendix. From the perspective of the generator building the input, the transitivity system defines classes of verbs according to their meaning and syntactic behavior (which often coincide, as is hypothesized in (Levin, 1993) for example).

While systemic linguists have developed complex transitivity systems to account for the structure of the clause, lexicalist approaches to grammar have developed the notion of subcategorization. The idea is that in any phrase, the lexical head determines the structure and order of the subconstituents — or dependents in a dependency-based grammar like Meaning Text Theory (Mel'cuk and Pertsov, 1987b) — of the phrase. To capture this dependency, HPSG grammars describe each lexical item capable of governing arguments with a feature called the subcategorization list, called *subcat*. The subcat of a verb is a list of the constituents the verb expects to form a complete phrase. So for example, the verb *to deal* as in *"AI deals with logic"* is represented in the lexicon with a subcat $deal[Subcat(< NP >< PP : with >)]$ which indicates that to produce a clause headed by this verb, one must find an NP and a PP headed by the preposition *with*. In HPSG, the subcat is also related to the semantics of the verb and the semantic function of its arguments as in the following example:

$$deal \left[\begin{array}{ll} \text{CAT} & \left[\begin{array}{ll} \text{HEAD} & \text{verb} \\ \text{SUBCAT} & < \boxed{1} NP[nom], \boxed{2} PP[P : with] > \end{array}\right] \\ \text{CONTENT} & \left[\begin{array}{ll} \text{RELN} & \text{has-topic} \\ \text{DOMAIN} & \boxed{1} \\ \text{TOPIC} & \boxed{2} \end{array}\right] \end{array}\right]$$

Every linguistic constituent in HPSG is associated to a feature structure containing a feature CONTENT describing the information contents realized by the linguistic element. In the example, the verb realizes a conceptual relation `has-topic(domain, topic)`.

Thus the subcategorization approach provides roughly the same type of information as the transitivity system in systemic grammars: mapping from semantic roles to syntactic arguments, determination of the syntactic category of each argument and of the order of the arguments. The differences are that the subcat feature describes a single lexical entry whereas the transitivity approach aims to describe general classes of semantically similar verbs and that the HPSG approach associates this information with a lexical item whereas the systemic approach associates it with a semantic input.[5]

The two scenarios are supported by SURGE: we view a process type as an abbreviation of the specification of a lexical entry for the main verb. A program preparing an input for SURGE can either find a process type corresponding to the semantic relation to be expressed in Table 1 or Table 2 in Appendix, or alternatively choose a lexical entry in a lexicon with all its subcat information specified. In this case, the process type is entered as "lexical" and the process must provide a subcat feature.

The two alternative options are shown in Fig 1 for a process specified in terms of transitivity, and in Fig 15 for a lexical process. When a process is specified using transitivity, the arguments are entered under the feature "participants" whereas the feature "lex-roles" is used for lexical processes.

At this stage, the input specification looks like the following template (where the set of participants depends on the process type):

$$\left[\begin{array}{ll} \text{cat} & \text{clause} \\ \text{process} & \left[\begin{array}{ll} \text{type} & < \text{type} > \\ \text{lex} & < \text{verb} > \\ \text{tense} & < \text{tense} > \\ \text{polarity} & < \text{polarity} > \end{array}\right] \\ \text{participants} & \left[\begin{array}{ll} \text{agent} & ... \\ \text{affected} & ... \\ ... & \end{array}\right] \\ ... & \end{array}\right]$$

5 HPSG does provide the possibility to build sortal hierarchies of relations (Pollard and Sag, 1994, p.287). In our approach, this type of generalization would then be captured in the lexical chooser and not in the syntactic realization module.

or for lexical roles:

$$\left[\begin{array}{ll} \text{cat} & \text{clause} \\ \text{process} & \left[\begin{array}{ll} \text{type} & < \text{type} > \\ \text{lex} & < \text{verb} > \\ \text{tense} & < \text{tense} > \\ \text{polarity} & < \text{polarity} > \end{array}\right] \\ \text{lex-roles} & \left[\begin{array}{ll} < \text{role1} > & \ldots \\ < \text{role2} > & \ldots \\ \ldots & \end{array}\right] \\ \ldots & \end{array}\right]$$

Once the participants or lex-roles are specified in the clause pattern, the kernel predicate–argument relation can be extended by merging in additional information, which surfaces as adjuncts or disjuncts[6]. This stage is in general performed by a *sentence planner* as discussed in (Robin, 1994). From the point of view of the generator building an input, adjuncts and disjuncts can be inserted onto clauses using the relations listed in Table 4, Table 5 and Table 6 in Appendix. For each relation, the table also lists which category of filler it accepts (clause, PP or NP).

In summary, the ideational dimension of the clause is specified by providing the features `process`, `participants` (or `lex-roles`) and an appropriate list of adjunct and disjuncts labelled functionally.

The inter-personal dimension is encoded in the feature `mood` of the clause which is described in Table 3 in Appendix. The textual dimension is encoded using the `focus` feature which is only partially supported in the current version, but see below the use made of it by the KNIGHT system.

4.2 Interface with the Underlying Ontology

For a generator, the decision to use lexical processes vs. the transitivity abbreviations corresponds largely to the question whether to use an "upper model" as the basis for the correspondance between the generator's ontology and the syntactic realization component (Bateman, 1989). An upper-model is often presented as the most general part of an ontology which is motivated by linguistic generalizations. When using an upper-model, the scenario is the following: the application constructs its specific ontology as a specialization of the upper-model; since domain-specific concepts are specializations of linguistic terms, the mapping from them to lexical items is made easier. For example, if the upper model includes the notions of the transitivity system, domain relations will be categorized in the application knowledge representation as specializations of `material` or `possessive` relations. The mapping from such a domain relation to the corresponding SURGE input is thus direct.

In contrast, if one does not build the application's domain ontology and the syntactic process classification on the same most general categories, then another mechanism must be used to bridge the gap between domain relations and process descriptions as expected by SURGE. In this case, a lexical chooser must be used to map a domain relation to the most appropriate lexical item.

While the upper-model approach is attractive by its simplicity, we have found that ofen generators cannot exploit it for two main reasons: the first one is that generators can be built on an existing domain ontology, which has been motivated by non-linguistic factors. In such a case, domain relations are not specializations of any linguistic categories. The second reason is related to the notion of "metaphorical expression" discussed in (Halliday, 1994, Chap.10). Consider the following example: in the basketball domain,

6 The difference between participants and adjuncts is discussed in Sect.5.2.

one of the central domain relations is that relating a player to a performance. It is encoded as:

```
performance(player, statistics)
```

This single domain relation can be expressed in the following distinct realizations:

- *Jordan scored 36 points* / material
- *Jordan had 36 points* / possessive
- *Jordan's 36 points* / possessive nominal
- *Jordan led the Bulls to victory with 36 points* / means adjunct
- *Jordan finished with 36 points* / locative(?)

In this example, a single domain predicate can be mapped to relations of different types from the perspective of the transitivity system. It would be difficult to locate such a predicate under an upper-model where the relations of `material` and `possessive` are disjoint.

The alternative mode of mapping from domain relation to SURGE process specification is using a lexical chooser which maps directly from domain relations (possibly organized in a taxonomy to capture generalities within the domain) to all desired linguistic realizations.

Note that the two options are not incompatible and can be easily combined. The generator preparing inputs can prepare a process type when found, which produces then a short SURGE specification, and the rest of the time rely on an explicit mapping from domain predicate to lexical process.

In summary, we find that the transitivity system provides a useful abstraction to mediate between a specific domain ontology and syntax. It is a useful grammatical generalization but there is little benefit in viewing it as an upper model in the sense that it would be more generic than the domain ontology. In many cases, forcing such a generic/specific relation between the grammatical specification and the domain ontology would have as effect a reduction in paraphrasing power.

4.3 How to Prepare Input Specifications

Client programs can use different techniques to build input specifications for SURGE. The easiest approach is to develop a set of FD templates to be filled by the client program in a procedural manner. This approach is illustrated by the KNIGHT system (Lester, 1994a). A more complex approach is to develop a declarative lexical chooser to map domain descriptions and communicative goals to inputs. This is illustrated in the ADVISOR and STREAK systems and is detailed in (Elhadad, McKeown, and Robin, 1997).

FD Templates. The first approach to build input specifications is to use FD templates, as used in KNIGHT. The KNIGHT system is used to explain entities in a large-scale knowledge base in the biology domain. The focus of the work was on the planning of coherent paragraph-length explanations, with intelligent selection of the level of detail, of the content of the explanation based on didactic criteria and on the overall robustness of the system. It included an extensive evaluation phase demonstrating the fluency and appropriateness of the generated paragraphs by comparing them with paragraphs written by domain experts. KNIGHT uses SURGE as a syntactic realization component. The interface between the content planner and SURGE is called the "functional description writer." The FD writer relies on a library of FD templates. Its main functions given an input knowledge base description for an event or relation are: to extract an appropriate FD template

$$
\begin{bmatrix}
\text{cat} & \text{clause} \\
\text{process} & \begin{bmatrix} \text{type} & \textit{material} \\ \text{verb} & < \text{verb-string} > \end{bmatrix} \\
\text{participants} & \begin{bmatrix} \text{agent} & < \text{transporter-fd} > \\ \text{affected} & < \text{transported-entities-fd} > \end{bmatrix} \\
\text{pred-modif} & \begin{bmatrix} \text{origin} & \begin{bmatrix} \text{cat} & \text{pp} \\ \text{np} & < \text{source-fd} > \end{bmatrix} \\ \text{destination} & \begin{bmatrix} \text{cat} & \text{pp} \\ \text{np} & < \text{destination-fd} > \end{bmatrix} \\ \text{path} & \begin{bmatrix} \text{cat} & \text{pp} \\ \text{np} & < \text{conduit-fd} > \end{bmatrix} \end{bmatrix} \\
\text{circum} & \begin{bmatrix} \text{time} & < \text{process-fd} > \end{bmatrix}
\end{bmatrix}
$$

Figure 8
FD template for the domain-specific `transportation` event

from the FD library, to map knowledge base components to the slots of the template and finally to combine several FDs into a fluent paragraph, using post-processing heuristics.

The FD library contains three types of templates:

- A lexicon to map simple entities to nominal FDs.
- Generic clause templates to describe generic relations among KB entities such as kind-of or part-of.
- Specific clause templates for specific domain relations such as "maturation", "reproduction" or "transportation".

Figure 8 shows the form of a domain-specific template used to map an instance of the "transportation" concept to a clause. This template can be used to generate a sentence such as: *"During pollen grain transfer, a pollen grain is transferred from the anther to the stigma."* Each slot in the template corresponds to information available in the KB description. The FD writer includes procedures to recursively map each slot to the embedded FD.

Noun phrase templates are stored in the lexicon as FD templates as well. In fact, the lexicon is folded into the knowledge base, as the FD templates are attached to the KB descriptions as annotations. NP construction can involve the construction of conjunctions and the embedding of relative clauses as modifiers. In the NP construction process, each KB slot is mapped to a specific type of modifier (describer, PP qualifier or relative qualifier).

Once a basic sequence of FD templates is built, the FD writer applies a set of post-processing heuristics to polish the output and make the output paragraphs more fluent. At this stage, the advantage of using a syntactic realization engine like SURGE with explicit input specifications instead of string templates becomes critical. The optimizations include:

- **Focus tracking**: the discourse model implemented in KNIGHT keeps track of which entities are introduced as new entities or remain as continuation topic in a subsequent sentence. Accordingly, the FD writer includes `focus` annotations in the FDs. Thus the same template of `reproduction` can be used to generate: "*During megasporogenesis, a megaspore mother cell produces four megaspores*" with focus on `parent` or "*Four megaspores are produced during megasporogenesis*" with focus on the `offspring` slot.

$$\begin{bmatrix} \text{semr} & \begin{bmatrix} \text{relation1} & \begin{bmatrix} \text{name} & < rel_1 > \\ \text{arguments} & \begin{bmatrix} 1 & < arg_{11} > \\ 2 & < arg_{12} > \end{bmatrix} \end{bmatrix} \\ \text{relation2} & \begin{bmatrix} \text{name} & < rel_2 > \\ \text{arguments} & \begin{bmatrix} 1 & < arg_{21} > \\ 2 & < arg_{22} > \end{bmatrix} \end{bmatrix} \\ \ldots & \end{bmatrix} \end{bmatrix}$$

Figure 9
Encoding of domain information in FDs as input to a FUF-based lexical chooser

- **Lexical Redundancy Checking**: the discourse planner can include legitimate predicates which sometimes give rise to redundant sentences such as "*the plant consists of plant tissue*" or "*microsporogenesis is a kind of sporogenesis*". Such trivial sentences are filtered out by comparing the lexical realization of the subject and object constituents. Such a test would be difficult to perform if the syntactic structure of the clause was not available.

- **Pronominalization**: a simple pronominalization algorithm has been developed to avoid repetitions of NPs across sentences. The algorithm relies on the complete constituent structure of the sentences to identify complete NPs and their heads.

FD templates are easy to implement and can yield good results thanks to optimizations. They are, however, not very compositional, as whole clause templates are specified all at once, and no mechanism exists to enrich clause templates with adjuncts. They are the preferred method when the level of paraphrasing required in the application is low.

Compositional Lexical Chooser. A more compositional approach is used in the ADVISOR system (Elhadad, McKeown, and Robin, 1997). In this approach, the client program interfaces with a declarative lexical chooser, itself written in FUF. The main focus here is on providing rich paraphrasing power in the lexical chooser. The client program first formats its content specification into FDs of the format shown in Fig.9.

In this format, domain information is encoded under a `semr` (for semantic representation) feature. We assume here that the information is in the form of binary relations. The lexical chooser performs then two types of decisions:

- **Syntagmatic**: if several relations are grouped in a single input, the lexical chooser determines which one is mapped to the process of the main clause, and which one is mapped to an adjunct or to a modifier of one of the constituents.

- **Paradigmatic**: for each relation and domain entity, a lexical entry must be selected among several candidates.

An example of lexical entry is shown in Fig.10 for the semantic relation `class_assignt` which holds between a class and its assignments in the ADVISOR domain. The lexical chooser proceeds as follows:

- A syntactic category for the candidate input is selected. This is often preselected by higher level decisions by the subcategorization constraint of the head constituent. At the toplevel, this decision is made by the "sentence planner".

% For each type of class_relation, determine possible
% lexicalization and its cat and identify sub-constituents

$$
\left[\begin{array}{ll}
semr & \left[\begin{array}{ll} relation_1 & \left[\begin{array}{ll} name & class\_assignt \\ class & [1] \\ assignments & [2] \end{array}\right] \end{array}\right] \\
process & \left[\begin{array}{ll} semr & \left[\begin{array}{ll} name & class\_assignt \end{array}\right] \end{array}\right] \\
 & \\
\multicolumn{2}{l}{\left\{\begin{array}{l}
\text{ALT ASSIGNMENTS} \\
\text{\% Lexicalizations: C requires A, C has A} \\
\text{\% there is A in C, in C they do A} \\
 \\
\text{\% Branch 1: C has A} \\
\left[\begin{array}{ll}
process & \left[\begin{array}{ll} type & possessive \\ lex & \text{“}have\text{''} \end{array}\right] \\
 & \text{\% lex\_cset declaration to handle recursion} \\
lex\_cset & \langle\ [3]\quad [4]\ \rangle \\
participants & \left[\begin{array}{ll} possessor & [3]\left[\begin{array}{ll} semr & [1] \end{array}\right] \\ possessed & [4]\left[\begin{array}{ll} semr & [2] \end{array}\right] \end{array}\right]
\end{array}\right] \\
\text{...other branches...}
\end{array}\right\}}
\end{array}\right]
$$

Figure 10
Fragment of a compositional lexicon

- The lexicon is unified with the input FD and, based on the value of the `semr` feature, a lexical entry is selected. For example, if the input contains the `class_assignt` relation, the lexical entry shown in Fig.10 would be selected. This entry encodes the possible lexicalizations of the semantic relation as a disjunction (denoted by the curly braces around a set of alternative FDs). The lexical chooser non-deterministically selects one possible lexicalization. For example, the `class_assignt` relation can be mapped to a possessive process.
- The lexical entry instantiates the syntactic features of the head constituent of the current category (it fills the `process` feature), and indicates on which subconstituents to continue the recursion (with the `lex_cset` feature).
- For each subconstituent, the lexical entry indicates under the `semr` feature the part of the semantic formula that needs to be realized. For example, the `semr` feature of the `possessor` and `possessed` features are instantiated with sub-parts of the `semr` of the toplevel constituent.

The last step is the generation counterpart of a compositional interpretation algorithm. Overall, the lexical chooser generally builds the syntactic structure in top-down recursive fashion following the algorithm below:

1. Choose a concept to head the sentence structure.
2. Choose a verb lexicalizing this head concept.
3. Choose which other concepts to map onto each argument of this verb.
4. Choose a sentential adjunct for each remaining concept.

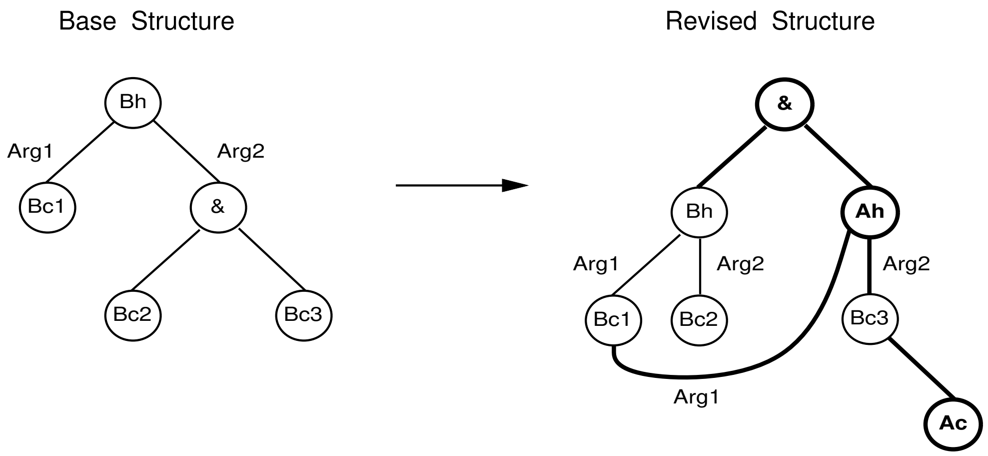

Figure 11
Example revision rule used in STREAK

5. Recursively choose (a) the syntactic category, (b) head word and (c) internal structure of each verb argument and sentential adjunct.

However, one of its main advantage is that it also allows a single element from the input semantic formula to be realized at many different ranks in the syntactic input specification, producing a rich paraphrasing power. The techniques used to implement such flexible lexicalization are detailed in (Elhadad, McKeown, and Robin, 1997). The compositional approach is prefered when complex paraphrasing is required.

Revision systems. A third technique to produce input specifications is illustrated by the STREAK system (Robin, 1994). The technique relies on revision to incrementally build a complex input specification by attaching new constituents to a base input FD. This is illustrated in the following sequence of increasingly complex sentences produced by STREAK (inserted constituents are shown in bold face):

1. *Dallas, TX – Charles Barkley registered 42 points Friday night as the Phoenix Suns routed the Dallas Mavericks 123 - 97.*

2. *Dallas, TX – Charles Barkley registered 42 points Friday night as the Phoenix Suns* **handed** *the Dallas Mavericks* **their 27th defeat in a row at home** *123 - 97.*

3. *Dallas, TX – Charles Barkley registered 42 points Friday night as the Phoenix Suns handed the Dallas Mavericks their* **franchise worst** *27th defeat in a row at home 123 - 97.*

4. *Dallas, TX – Charles Barkley* **matched his season record with** *42 points Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row at home 123 - 97.*

At each step, the client program finds opportunities to add syntactic information. The lexical chooser uses a rewriting system (implemented in FUF) to hook new constituents on existing input FDs. As the addition of new FDs on the base FD can significantly modify the structure of the base FD, the system is in general non monotonic. It uses a set of revision rules to extend the input FD, an example of which is shown in Fig.11. This

rule (the "coordination promotion schema") is used to produce the following increment:
`source`: *"Larry Smith added [[12 points] and [25 rebounds.]]"*
`target`: *"[[Larry Smith added 12 points] and [∅* **matched a season high** *25 rebounds.]]"*

Such revision rules are highly desirable to construct compositionally complex input specifications. Note that one of the contributions of SURGE to the system, was the possibility to even express such rules in a formal manner. The revision approach is prefered when the generator aims at producing syntactically complex sentences.

4.4 From skeletal thematic trees to English sentence: the Syntactic Realization Subtasks

The task of the realization component is to map the input specifications described in this section onto a natural language sentence. When an input specification is fully provided, the syntactic realization component must fulfill the following sub-tasks:

1. *Map thematic structure onto syntactic roles*: *e.g.,* `agent`, `process` and `created` onto `subject`, `verb-group` and `direct-object` (respectively) in I_1.
2. *Control syntactic paraphrasing and alternations*: for example, the selection of the passive for an embedded clause is discussed in Section 5.4.
3. *Prevent over-generation*
4. *Provide defaults for syntactic features*
5. *Propagate agreement features*, providing enough input to the morphology module
6. *Select function (closed-class) words*: *e.g., "she", "the"* and *"to"*.
7. *Provide linear precedence constraints* among syntactic constituents: *e.g.,* `subject` > `verb-group` > `direct-object` once the default active voice has been chosen.
8. *Inflect open-class words* (morphological processing): *e.g.,* the verb *"to score* as *"scored* in I_1.
9. *Linearize the syntactic tree into a string of inflected words* following the linear precedence constraints.
10. *Perform syntactic inference* as explained in Sect.5.4.

Having discussed how input specifications are constituted and highlighted the role of the syntactic realization component in their processing, we now turn our attention to how some of these tasks are implemented internally in SURGE.

5. How the Grammar Processes Inputs

5.1 Overall Organization

At the top-level, SURGE is organized into sub-grammars, one for each syntactic category: clause, nominals, determiner sequence, prepositional phrase, adjectival phrase and some specializations of them. Each sub-grammar encapsulates the relevant part of the grammar.

The category of an input constituent and of the corresponding sub-grammar is indicated by the special feature `cat`. The top-level input specification is thus first unified with SURGE. After this top-level unification is completed, the linguistic constituents in the input are identified. They are identified by a special feature in the grammar called `cset` (for constituent set). Each subconstituent is in turn recursively unified with the sub-grammar of the corresponding category.

For example, generating the sentence *"Michael Jordan scored two-thirds of his 36 points with 3 point shots"* (whose input was shown in Fig.3, page5) involves unifying:

1.The input FD as a whole with the clause sub-grammar.

2.The process FD with the verb group sub-grammar.

3.Both the agent and created participant sub-FDs with the nominal sub-grammar.

4.The instrument predicate modifier sub-FD with the PP sub-grammar.

As this process unfolds, the complete constituent structure of the sentence is built. This resulting constituent structure for I_2 is shown in Fig.12, with for each constituent, both its position in the syntactic structure and its original position in the input specification (with the functional labels). For example, the `participant|agent` constituent is mapped to the `oblique|1` and to the `synt-roles|subject` positions.

In addition, in SURGE, syntactic categories are hierachically classified and sub-grammars can be recursively divided into sub-sub-grammars. For example, the nominal sub-grammar is divided into sub-sub-grammars for pronouns, for NP headed by proper nouns, for NP headed by common nouns, etc.

Orthogonally, to this hierachical organization, each sub-grammar is also divided into a set of *systems* (in the systemic sense), each one encapsulating an essentially independent set of decisions, constraints and features.

In the rest of this section, we first briefly present some of the main systems implemented in SURGE and discuss some of the major implementation decisions for each one. Overall, SURGE is organized as a systemic grammar although internally it diverges from pure systemic principles towards a more structural representation. This internal implementation level allows us to integrate an account for phenomena like control and binding for which HPSG provides a compact and elegant theory without affecting the interface with the client program which remains largely functional.

5.2 The Clause Systems

At the clause level, SURGE input specification labels the linguistic constituents by the *thematic roles* they play in the clause semantics. Following (Halliday, 1994), the *thematic* roles accepted by SURGE in input clause specifications first divide into: *nuclear* and *satellite* roles. Nuclear roles, answer the questions "who/what was involved?" about the situation described by the clause. They include the semantic class of this situation, specified under the *process* feature generally surfacing as the `verb`[7] and its associated *participants* surfacing as verb arguments. In contrast, satellite roles (also called *adverbials*) answer the questions "when/where/why/how did it happen?" and surface as the remaining clause complements.

Semantically, nuclear roles are tightly associated with the type of process described by the clause, while satellites are versatile roles compatible with virtually any process type. Syntactically, participants can be distinguished from satellites in that[8]: (1) they

7 Or as `direct-object` in the case of clauses headed by support-verbs (Gross, 1984).
8 While each of these three tests has a number of exceptions, taken together they are very reliable.

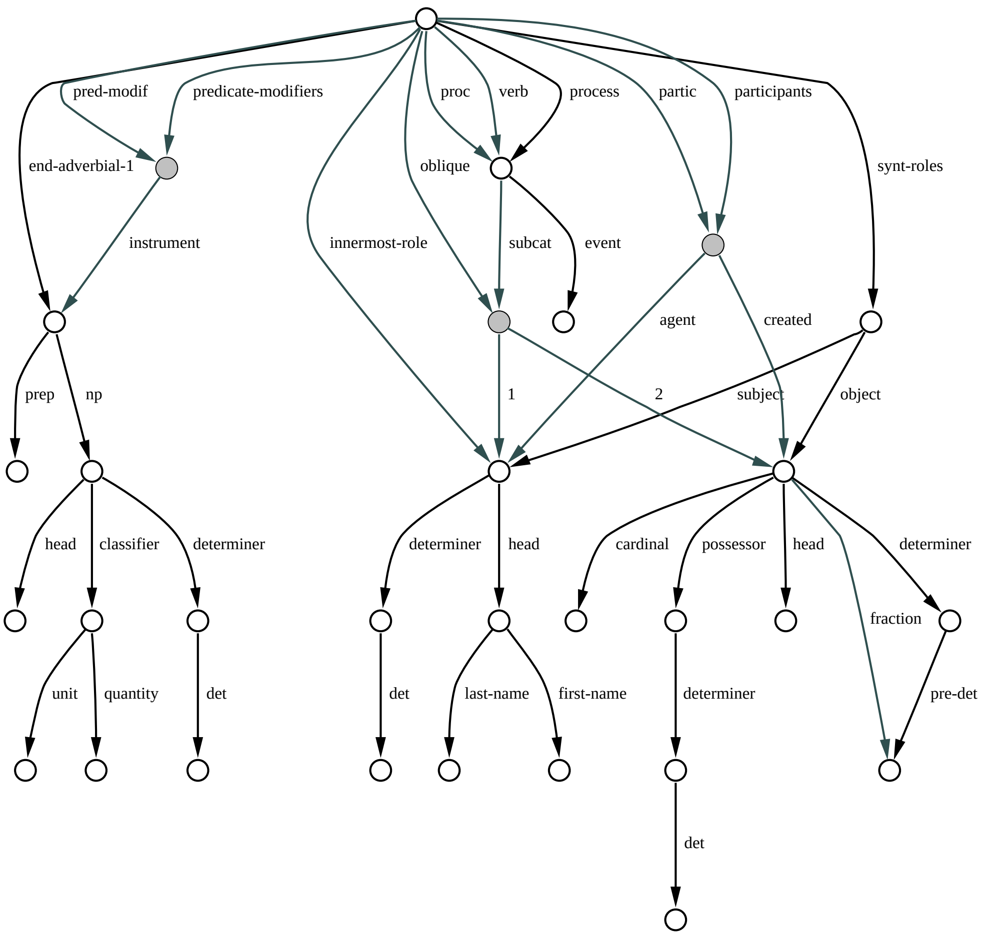

Figure 12
Constituent structure constructed by SURGE for Input I_2 of Fig.3:
"Michael Jordan scored two-thirds of his 36 points with 3 point shots"
The main constituent backbone is shown in black. Functional labels mapped on the syntactic constituents are shown in gray.

surface as `subject` for at least one syntactic alternation (2) they can neither be moved around in the clause nor (3) omitted from the clause while preserving its grammaticality.

Following this sub-division of thematic roles, the clause sub-grammar is divided into four orthogonal systems, as a standard systemic grammar:

- *Transitivity*, which handles mapping of nuclear thematic roles onto a default core syntactic structure for main assertive clauses.
- *Voice*, which handles departures from the default core syntactic structure triggered by the use of syntactic alternations (*e.g.,* passive or dative moves).
- *Mood*, which handles variations in terms of speech acts (*e.g.,* interrogative or imperative clause) and syntactic functions (*e.g.,* matrix *vs.* subordinate clause).
- *Adverbial*, which handles mapping of satellite roles onto the peripheral syntactic structure, their respective ordering and their integration into relatives and questions.

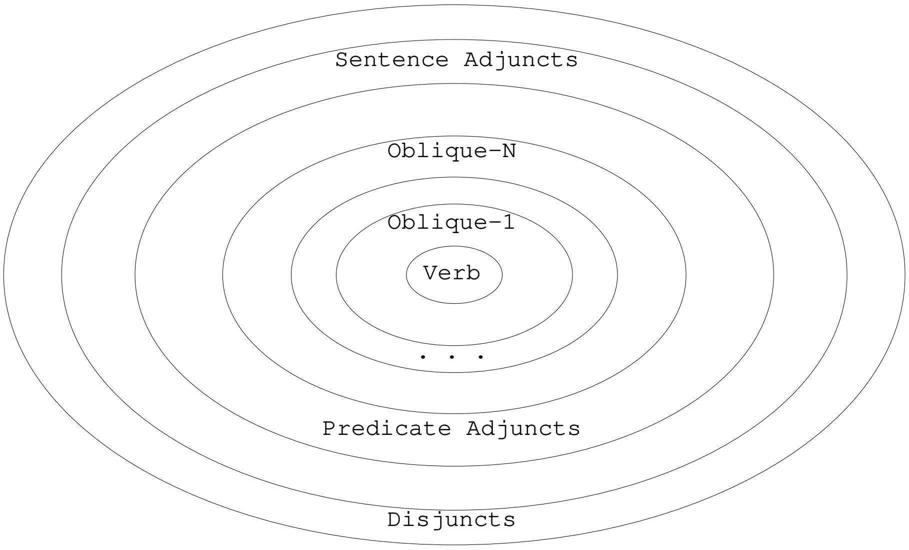

Figure 13
Nuclearity of clause constituents

The transitivity system is SURGE's interface to the semantic encoding used in the client program (in terms of process type and participant roles). In the implementation, the transitivity system introduces two levels of encoding for the nuclear roles: the functional participant roles are first mapped to a level we call the *oblique* hierarchy, following HPSG terminology. The oblique constituents are just numbered 1, 2, 3... according to their "proximity" to the process of the clause. The nuclearity of the various clause constituent classes is illustrated in Fig.13. In practice, the oblique level is an abstraction of the functional roles like agent, possessor etc. which allows a more compact encoding of the voice system. Second, the oblique level is mapped to a level we call *synt-roles* and which includes the features `subject, object, iobject, by-obj` and `dative-obj`. Note that the level of synt-roles is also a significant deviation from HPSG. In fact, it allows us to encode what is accounted for in HPSG by lexical rules in the grammar.

Note that when the input specification uses a lexical type process, the first mapping is simply skipped, as the lexical entry for the process directly maps functional roles to oblique positions.

The sub-tasks of the transitivity system are thus to:

- Define the acceptable combinations of participant roles and check whether they are compatible with the process type and main verb.
- Map participant roles onto obliques roles.
- Define the acceptable syntactic realizations of each participant.

This implementation of the transitivity system is an example of the theory integration methodology we have followed during the development of SURGE. We illustrate below some of the benefits of this approach on an example involving control.

The adverbial system illustrates another example where no single theory provided us with a ready-to-implement solution. It encodes a more subtly constrained mapping from thematic structures to syntactic roles than the transitivity system. While nuclear roles surface as mutually exclusive core syntactic functions (*e.g,* each clause can only have one subject and one direct object), satellite roles surface as one of three peripheral syntactic functions, *predicate adjuncts*, *sentence adjuncts* and *disjuncts* following (Quirk et al., 1985, pp. 504-505), with potentially multiple instances of each. The adjuncts *vs.* disjuncts distinction is purely syntactic and accounts for general alternations (*e.g.,* only adjuncts can be clefted or appear in alternative questions). The distinction between predicate and sentence adjunct is semantic: the former modifies only the *process* of the clause, while the latter elaborates on the entire situation described by the clause. It has subtle repercussions in terms of syntactic alternations, possible positions and co-occurrence (*e.g.,* predicate adjuncts cannot be fronted). SURGE reflects this distinction in input where satellite roles can be either `predicate-modifier` which get mapped onto predicate adjuncts and `circumstancials` which get mapped onto either sentence adjuncts or disjuncts (depending on semantic label, syntactic category and lexical content). The relative distance of different types of complements from the process is illustrated in Fig.13.

Integrating descriptions by (Quirk et al., 1985), (Thompson and Longacre, 1985) (Halliday, 1994), SURGE can currently map a large variety of adverbials with many different realizations. Tables 4, 5 and 6 in Appendix summarize the types of predicate adjuncts, sentential adjuncts and disjuncts which are available in SURGE.

5.3 Nominals

Nominals are an extremely versatile syntactic category, and except for limited cases (cf. (Vendler, 1968), (Levi, 1978), (Fries, 1970)), no linguistic semantic classification of nominals has been provided. Consequently, while for clauses, input can be provided in thematic form, for nominals it must be provided directly in terms of syntactic roles. The task of mapping domain-specific thematic relations to the syntactic slots in an NP is therefore left to the client program (*cf.* (Elhadad, 1996) for a discussion of this process).

SURGE distinguishes among the following nominal types: *common nouns, proper nouns, pronouns, measures, noun-compounds* and *partitives.* Noun-compounds can have a deep embedded structure. Measures have a specific syntactic behavior (when used as classifiers the head noun of the measure does not take a number inflection, *e.g., "a 3 meter boat" vs. "the boat measures 3 meters"*).

Some specific nominal-like categories have been added to SURGE to cover widely used data which is rarely described in grammars: these include person names, addresses (locations), dates and times. For these categories, we developed the grammar by simply attempting to cover a wide variety of data observed in various corpora.

The specific syntactic roles accepted by SURGE for nominal description are (given in their partial order of precedence): *determiner-sequence, describer, classifier, head,* and *qualifier.* The various syntactic structures covered in SURGE are illustrated in Table. 7 in Appendix.

The determiner-sequence is itself decomposed into the following elements:
pre-determiner, determiner, ordinal, cardinal, quantifier. This sub-grammar is special in that it is mainly a closed system, *i.e.,* all lexical differences must be determined by some configuration of syntactic features. The determiner sequence sub-grammar currently covers is illustrated in Fig.8. Unexpectedly, it is one of the most complex parts of the grammar in terms of implementation.

5.4 An Example of Complex Grammatical Processing: Control

One of the key advantages of functional unification as a computational formalism is that it allows the grammar writer to encode each system largely independently and hence to

$$\begin{bmatrix}
\text{cat} & \text{clause} \\
\text{process} & \begin{bmatrix} \text{type} & \text{material} \\ \text{effect-type} & \text{creative} \\ \text{lex} & \text{``score''} \\ \text{tense} & \text{past}\end{bmatrix} \\
\text{partic} & \begin{bmatrix} \text{agent} & \begin{bmatrix} \textbf{index} & [0] \\ \text{cat} & \text{proper} \\ \text{head} & \begin{bmatrix} \text{cat} & \text{person-name} \\ \text{first-name} & \begin{bmatrix}\text{lex} & \text{``Michael''}\end{bmatrix} \\ \text{last-name} & \begin{bmatrix}\text{lex} & \text{``Jordan''}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\
\text{created} & \begin{bmatrix} \text{cat} & \text{np} \\ \text{fraction} & \begin{bmatrix}\text{num} & 2 \\ \text{den} & 3\end{bmatrix} \\ \text{possessor} & \begin{bmatrix}\text{cat} & \text{personal-pronoun} \\ \text{gender} & \text{masculine}\end{bmatrix} \\ \text{cardinal} & \begin{bmatrix}\text{value} & 36\end{bmatrix} \\ \text{head} & \begin{bmatrix}\text{lex} & \text{``point''}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\
\text{pred-modif} & \begin{bmatrix}\text{instrument} & \begin{bmatrix}\text{np} & \begin{bmatrix}\text{classifier} & \begin{bmatrix}\text{cat} & \text{measure} \\ \text{quantity} & \begin{bmatrix}\text{value} & 3 \\ \text{digit} & \text{yes}\end{bmatrix} \\ \text{unit} & \begin{bmatrix}\text{lex} & \text{``point''}\end{bmatrix}\end{bmatrix} \\ \text{number} & \text{plural} \\ \text{definite} & \text{no} \\ \text{head} & \begin{bmatrix}\text{lex} & \text{``shot''}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} \\
\text{circum} & \begin{bmatrix}\text{result} & [6]\end{bmatrix}
\end{bmatrix}$$

Figure 14
Input specification I_4 for the sentence S_4 *"Michael Jordan scored two-thirds of his 36 points with 3 point shots, to enable, Chicago to hand New York a season high sixth straight loss."*

successively focus on limited aspects of syntax at knowledge acquisition time. Accounts of complex syntactic phenomenon which simultaneously constrain several systems then naturally emerge at run-time, from the unification of the features that these systems share. We illustrate this point by showing how the three clause systems that we just mentioned, transitivity, voice and mood, cooperate through unification to account for (1) subject control and (2) perform complex syntactic inference.

Consider the example sentence S_4, whose input specification I_4 is shown in Fig.14 and Fig.15:
"Michael Jordan scored two-thirds of his 36 points with 3 point shots, to enable Chicago to hand New York a season high sixth straight loss."

As in the case of many concise complex sentences, S_4 contains two implicit recoverable semantic constituents, namely the logical subjects of the two embedded infinitive clauses. These constituents do not need to explicitly surface in the syntactic structure, because they are semantically co-indexed with constituents of the matrix clause. Lack of such implicitation would lead to awkwardly repetitive and misleading sentences such as:
? *"Michael Jordan scored two-thirds of his 36 points with 3 point shots,* $<$ *for Michael Jordan* $>$ *to enable Chicago* $<$ *for Chicago* $>$ *to hand New York a season high sixth straight loss."*

The SURGE input specification I_4 for S_4 does not indicate whether any constituent should be left implicit. It only indicates semantic co-reference relations between the fillers of different thematic roles – by way of shared `index` feature.

There are two shared indexes in I_4. The first, `index [0]` indicates that the `influence` of the `result` clause is co-referent with the `agent` of the main clause (Michael Jordan).

$$[6] = \left[\begin{array}{ll}
\text{cat} & \text{clause} \\
\text{mood} & \text{to-infinitive} \\
\text{process} & \left[\begin{array}{ll}
\text{type} & \text{lexical} \\
\text{lex} & \text{“enable”} \\
\textbf{subcat} & \left[\begin{array}{ll}
\mathbf{1} & \mathbf{[1]} \\
\mathbf{2} & \mathbf{[2]} \left[\begin{array}{ll} \textbf{index} & \mathbf{[4]} \\ \textbf{cat} & \textbf{np} \end{array}\right] \\
\mathbf{3} & \mathbf{[3]} \left[\begin{array}{ll} \textbf{cat} & \textbf{clause} \\ \textbf{mood} & \textbf{to-infinitive} \\ \textbf{synt-roles} & \left[\ \textbf{subject}\ \left[\ \textbf{index}\ \ \mathbf{[4]}\ \right]\ \right] \end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{lex-roles} & \left[\begin{array}{lll}
\text{influence} & [1] & \left[\begin{array}{ll} \text{cat} & \text{proper} \\ \text{head} & \left[\begin{array}{ll} \textbf{index} & \mathbf{[0]} \\ \ldots & \end{array}\right] \end{array}\right] \\
\text{influenced} & [2] & \left[\begin{array}{ll} \textbf{index} & \mathbf{[4]} \\ \text{cat} & \text{proper} \\ \text{lex} & \text{“Chicago”} \end{array}\right] \\
\text{soa} & [3] &
\end{array}\right]
\end{array}\right]$$

$$[3] = \left[\begin{array}{ll}
\text{cat} & \text{clause} \\
\text{process} & \left[\begin{array}{ll} \text{type} & \text{composite} \\ \text{relation-type} & \text{possessive} \\ \text{lex} & \text{“hand”} \end{array}\right] \\
\text{partic} & \left[\begin{array}{ll}
\text{agent} & \left[\begin{array}{ll} \textbf{index} & \mathbf{[4]} \\ \ldots & \end{array}\right] \\
\text{affected} & [5] \left[\begin{array}{ll} \text{cat} & \text{proper} \\ \text{lex} & \text{“New York”} \end{array}\right] \\
\text{possessor} & [5] \\
\text{possessed} & \left[\begin{array}{ll}
\text{cat} & \text{common} \\
\text{definite} & \text{no} \\
\text{classifier} & \left[\begin{array}{ll} \text{cat} & \text{noun-compound} \\ \text{classifier} & \left[\ \text{lex}\ \ \text{“season”}\ \right] \\ \text{head} & \left[\ \text{lex}\ \ \text{“high”}\ \right] \end{array}\right] \\
\text{head} & \left[\begin{array}{ll}
\text{cat} & \text{measure} \\
\text{quantity} & \left[\begin{array}{ll} \text{cat} & \text{compound-ordinal} \\ \text{numeral} & \left[\begin{array}{ll} \text{cat} & \text{ordinal} \\ \text{value} & 6 \end{array}\right] \\ \text{complement} & \left[\ \text{lex}\ \ \text{“straight”}\ \right] \end{array}\right] \\
\text{unit} & \left[\ \text{lex}\ \ \text{“loss”}\ \right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 15
Input specification for the result clause C_4 *“to enable Chicago to hand New York a season high sixth straight loss”* of sentence S_4

SURGE leaves the `influence` role implicit in the output sentence S_4 as the result of the following thread of decisions:

1. `influence` gets mapped onto the `oblique|1` role from the explicit subcategorization specification of the `enable` process in the input.
2. `oblique|1` then gets mapped onto `subject` as a result of SURGE choosing the default active voice.
3. The adverbial system implements a rule in which the `result` clause is mapped to a to-infinitive clause without subject when its subject is co-indexed with the agent of the matrix clause.

In fact, the to-infinitive mood is not selected directly, instead, the adverbial system simply indicates that the subject of the result clause should not be expressed when it is co-indexed with the agent of the matrix. The mood system in turn determines that a to-infinitive mood must be used, since any other possible mood would require an explicit subject (as shown in Table 6).

Such complex interactions among systems are easily implemented in SURGE thanks to the bidirectionality of the underlying FUF formalism.

An even more complex inference procedure is demonstrated in Fig.16. Let us first examine the second shared index feature in I_4, `index [4]`. The co-indexing indicates that the influenced role of the result clause corefers with the agent of the soa (state-of-affair) embedded clause in S_4:

"... to enable Chicago **to hand New York a season high sixth straight loss."**

The process *"enable"* exhibits here the behavior of an object-control verb (Pollard and Sag, 1994, p.286). In the clause:

Jordan enables Chicago to win

the object of *enable* is also the subject of *to win*. In contrast, with a subject-control verb like *promise*: *Jordan promised Chicago to come back*, the subject of the matrix clause is also the subject of the embedded clause. This behavior is lexical (small classes of verbs with common semantic properties exhibit a similar behavior). We therefore encode it in the lexical entry for the process, using a lexical type specification with a `subcat` feature. For the verb *enable*, the control behavior is completely encoded in the following FD, which comes as is from the lexicon:

$$\left[\begin{array}{ll}
\text{type} & \text{lexical} \\
\text{lex} & \text{“enable”} \\
\text{subcat} & \left[\begin{array}{lll}
1 & [1] & \\
2 & [2] & \left[\begin{array}{ll} \text{index} & [4] \\ \text{cat} & \text{np} \end{array}\right] \\
3 & [3] & \left[\begin{array}{ll} \text{cat} & \text{clause} \\ \text{mood} & \text{to-infinitive} \\ \text{synt-roles} & \left[\begin{array}{ll} \text{subject} & \left[\begin{array}{ll} \text{index} & [4] \end{array}\right] \end{array}\right] \end{array}\right]
\end{array}\right]
\end{array}\right]$$

The co-indexing of [4] as the oblique role 2 of the verb and the subject of the embedded clause determines the control relation. The control principle itself is implemented in the mood system of SURGE.

Note that the control relation holds between a semantic role in the matrix clause (`oblique|2` which is lexically mapped to the `influenced` role) and a syntactic role in the embedded clause (`subject`). This fact is exploited in sentence S_4^b below, where the soa clause embedded in the same matrix, surfaces at the passive voice: *"Michael Jordan scored two-thirds of his 36 points with 3 point shots to enable a season high sixth straight loss* **to be handed by Chicago to New York**.

In terms of final inflected word string, the result clause C_4^b of S_4^b drastically differs from the corresponding result clause C_4 of S_4 above. However, as shown in Figs. 15 above and 16 below, in terms of SURGE input specification, these two clauses differ only in that the filler of the influenced role has been swapped with the possessed role of the soa role. Such contrast illustrates the abstraction power of SURGE's input specification.

Let us step through SURGE's decision to use the passive voice in C_4^b:

1. Because of the sub-categorization specification of *to enable*, the `subject` of the `soa` must be controlled by the `influenced`.

2. Because the `possessed` role of the `soa` is co-indexed with the `influenced` role, `possessed` must be mapped to the `subject` of the `soa`.

3. The voice system determines that the passive voice must be used to make of the `possessed` role the `subject` of a composite process of type material-possessive.

```
[ cat        clause
  mood       to-infinitive
  process    [ type    lexical
               lex     "enable"
               tense   past
               subcat  [ 1  [1]
                         2  [2] [ index  [4]
                                  cat    np ]
                         3  [3] [ cat         clause
                                  mood        to-infinitive
                                  synt-roles  [ subject [ index [4] ] ] ] ] ]
  lex-roles  [ influence   [1] [ cat   proper
                                 head  [ cat         person-name
                                         first-name  [ lex "Michael" ]
                                         last-name   [ lex "Jordan" ] ] ]
               influenced  [2] [ index       [4]
                                 cat         common
                                 definite    no
                                 classifier  [ cat         noun-compound
                                               classifier  [ lex "season" ]
                                               head        [ lex "high" ] ]
                                 head        [ cat       measure
                                               quantity  [ cat         compound-ordinal
                                                           numeral     [ cat    ordinal
                                                                         value  6 ]
                                                           complement  [ lex "straight" ] ]
                                               unit      [ lex "loss" ] ] ]
               soa         [3] [ cat      clause
                                 process  [ type           composite
                                            agentless      no
                                            relation-type  possessive
                                            lex            "hand" ]
                                 partic   [ agent      [ cat  proper
                                                         lex  "Chicago" ]
                                            affected   [5] [ cat  proper
                                                             lex  "New York" ]
                                            possessor  [5]
                                            possessed  [ index  [4]
                                                         ... ] ] ] ] ]
```

Figure 16
Input specification I_5 for the result clause C_5 *"... to enable a season high sixth straight loss to be handed by Chicago to New York"*
.

This complex example also justifies the introduction of the oblique level and the integration of lexical processes in SURGE in addition to the systemic transitivity systems.

5.5 Complex Processing: Binding

As a final example of both the type of complex processing performed by SURGE and the benefits of introducing HPSG elements in the systemic core of SURGE, consider the case of obligatory syntactic pronominalization shown in Fig.17. The binding theory of HPSG (Pollard and Sag, 1994, Chap.6) accounts for the contrast between the following sentences, where subscripts indicate co-reference:

- (6) "$Claudia_i$ likes $Claudia_j$."

$$I_6 = \begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \begin{bmatrix} \text{type} & \text{mental} \\ \text{lex} & \text{``like''} \end{bmatrix} \\ \text{partic} & \begin{bmatrix} \text{processor} & \begin{bmatrix} \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \\ \text{phenomenon} & \begin{bmatrix} \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$\Longrightarrow$"$Claudia_i$ likes $Claudia_j$."

$$I_7 = \begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \begin{bmatrix} \text{type} & \text{mental} \\ \text{lex} & \text{``like''} \end{bmatrix} \\ \text{partic} & \begin{bmatrix} \text{processor} & \begin{bmatrix} \textbf{index} & \textbf{[1]} \\ \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \\ \text{phenomenon} & \begin{bmatrix} \textbf{index} & \textbf{[1]} \\ \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$\Longrightarrow$"$Claudia_i$ likes $herself_i$."

$$I_8 = \begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \begin{bmatrix} \text{type} & \text{mental} \\ \text{lex} & \text{``think''} \end{bmatrix} \\ \text{partic} & \begin{bmatrix} \text{processor} & \begin{bmatrix} \textbf{index} & \textbf{[1]} \\ \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \\ \text{phenomenon} & \begin{bmatrix} \text{cat} & \text{clause} \\ \text{process} & \begin{bmatrix} \text{type} & \text{mental} \\ \text{lex} & \text{``like''} \end{bmatrix} \\ \text{partic} & \begin{bmatrix} \text{processor} & \begin{bmatrix} \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \\ \text{phenomenon} & \begin{bmatrix} \textbf{index} & \textbf{[1]} \\ \text{cat} & \text{proper} \\ \text{lex} & \text{``Claudia''} \\ \text{gender} & \text{feminine} \\ \text{animate} & \text{yes} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$\Longrightarrow$"$Claudia_i$ thinks that $Claudia_k$ likes her_i."

Figure 17
Contrasting input specifications for binding cases

- (7) "$Claudia_i$ likes $herself_i$."
- (8) "$Claudia_i$ thinks that $Claudia_k$ likes her_i."

We have implemented HPSG's binding theory in SURGE, through the following decisions:

- Because binding relations depend partly on surface structure, the binding relations are only computed at the end of the whole unification process, by the linearizer module.
- Because we want to relieve the input specification from the need to plan in advance whether a constituent could be turned into an anaphor or reflexive pronoun, we specify that all NP constituents can be "turned" into pronouns by adding a feature independent of their `cat` feature. This strongly indicates that this form of pronominalization is different from the discourse based pronominalization that would be performed by the content planner.

The implementation of the binding theory is partly procedural (it is implemented in the linearizer module). It is one of the limits of the underlying FUF formalism that it cannot express declaratively structural relations among constituents (such as the o-command relation defined in HPSG).

In summary, we have illustrated in this section the type of complex processing performed by SURGE. In terms of implementation, these examples illustrate how distinct systems interact through the use of shared features. In terms of development methodology, they justify a sort of hybrid approach to grammar development, where different aspects of distinct theories are integrated into one coherent framework.

6. Development Environment

One of the main lessons learned during the development of SURGE is that maintaining a large grammar is complex. We have tried to develop tools to assist in this task, but it remains difficult to learn to debug a large grammar and complex input specifications.

The tools we have developed to help use SURGE practically include the following:

- Visualization tools: graphs of input specifications and of constituent structures as shown in Fig.2 and Fig.12 have been generated by these automatic tools. Similarly, all the LaTeX encoding of feature structures shown in the paper have been produced automatically by the system. Finally, the feature type hierarchies can also be visualized graphically.
- Tracing tools: trace messages at varying levels of detail can be output. It remains a challenging issue to identify the real cause of failure when debugging SURGE inputs and traces tend to be too verbose.
- Large example set: several hundred example input specifications covering most of the features of SURGE are provided with the system. Cut and paste of examples remains one the most efficient methods of input preparation.

An important aspect of the usability of the system is its efficiency: in general, a complex SURGE input specification such as I_4 is processed in about 10 seconds on a 1995-level Sparcstation under Common Lisp. We are working on a new implementation of the FUF interpreter which should improve performance by an order of magnitude.

7. Related Work

Three other available systems provide reusable surface realization components: MUMBLE (Meteer et al., 1987), KPML (Bateman, 1996) and the systems developed at Cogentex (e.g., (Iordanskaja et al., 1994)). These three systems differ from SURGE in that they are all developed within a single linguistic theory (TAGs for MUMBLE, systemic functional for KPML and Meaning-Text Theory (MTT) (Mel'cuk and Pertsov, 1987b) for Cogentex's systems), whereas SURGE integrates systemic, HPSG and theory neutral encodings. Each system also puts the emphasis on different dimensions resulting in different strengths and weaknesses.

MUMBLE's determinism (motivated on cognitive grounds) makes it very efficient. Another of its strengths is the regular input provided by Meteer's text-structure (Meteer, 1992). KPML's strengths are comprehensive coverage, multi-linguality, an advanced development environment and encapsulation of syntactic knowledge through the inquiry mechanism. However, both of these systems seem to have a rather fixed order of decision making.

In addition, NIGEL (the core of KPML's English grammar) also has a tendency to over-generate. This last weakness has however recently been turned into a strength by the addition of a post-processor which filters outputs using a statistical language model (Knight and Hatzivassiloglou, 1995). This post-processor allows NIGEL's output to nicely degrade in the face of ill-formed or incomplete input, making it more robust than other systems.

MTT-based syntactic realization components are strong on extensibility (modular and declarative PROLOG implementation) and comprehensive coverage. But the fact that they must work in tandem with an MTT-based lexicon (the Explanatory Combinatorial Dictionary) makes them weak on versatility and encapsulation of syntactic knowledge.

SURGE's strengths are comprehensive coverage, encapsulation of syntactic knowledge, the extensibility of its purely declarative implementation and the versatility of its compact, and regular input where individual words and canned phrases can co-exist.

8. Conclusion

A re-usable syntactic realization component (or grammar for short) for language generation should thrive to satisfy the following properties:

1. *Comprehensive coverage*: it should be able to generate a large set of valid syntactic forms and to output many alternate syntactic forms from a single input.

2. *Prevention of over-generation*: it should generate only grammatical outputs (the counter-part of the previous criterion).

3. *Encapsulation of syntactic knowledge*: it should accept inputs that can be prepared by the client program or user with as little knowledge of syntax as possible.

4. *Regular input*: it should require inputs whose structure is well-defined and regular, and thus easily producible by a recursive client program.

5. *Partially canned input*: it should accept inputs where canned phrases can co-exist with individual words (Reiter, 1995), to be usable across the granularity gradient from simple template systems to fully compositional generation systems.

6. *Compact input*: it should provide appropriate defaults for all syntactic features, so that the input can contain only the few features that have non-default values.

7. *Extensibility*: it should be easy to add new syntactic constructs, and quickly verify their interaction with the rest of the grammar.

8. *Efficiency*: it should generate a sentence from the input specification as fast as possible.

9. *Robustness*: it should be able to generate grammatical fragments from ill-formed or incomplete inputs.

10. *User-friendliness*: it should be easy to use for computational linguists not familiar with the implementation details and easy to learn for users with little knowledge of linguistic theory.

11. *Versatility*: it should be usable in a wide variety of system architectures and application domains.

In this paper, we have tried to demonstrate how challenging addressing all of these desirable features remains. We have also reported on the main lessons learned from the development of SURGE:

- It is possible to develop a reusable syntactic realization component
- Syntactic realization can include complex processing and should rely on a flexible formalism to support it.
- Readiness to integrate results from different linguistic schools can help to address specific phenomena.
- Special attention needs to be paid to categories like dates and person names because they are poorly described in linguistics but are still central to many applications.
- Special attention must be paid to a sophisticated development environment for grammar development.

The development of SURGE continues, as prompted by the needs of new applications, and by our better understanding of the respective tasks of syntactic realization and lexical choice (Elhadad, McKeown, and Robin, 1997). We are specifically working on (1) integrating a more systematic implementation of Levin's alternations within the grammar, (2) extending composite processes to include mental and verbal ones using more recent results from Fawcett, (3) modifying the nominal grammar to support nominalizations and some forms of syntactic alternations (relying on (Fries, 1970)).

As it stands, SURGE provides a comprehensive syntactic realization component, easy to integrate within a wide range of architectures for complete generation systems. It is available on the WWW at `http://www.cs.bgu.ac.il/surge/`.

References

Abella, A. 1994. *From imagery to salience: locative expressions in context.* Ph.D. thesis, Computer Science Department, Columbia Universtity, New York, NY.

Allen, J.F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Bateman, J.A. 1989. Upper-modelling: current states of theory and practise - part1: foundations and related approaches -.

Bateman, John, 1996. *KPML Development Environment – Multilingual linguistic resource development and sentence generation.* German Centre for Information Technology (GMD), Institut fur integrierte Publikations und Informationssysteme (IPSI) – Project KOMET – Dolivostr. 15, Darmstadt, Germany. Available at http://www.darmstadt.gmd.de/publish/komet/kpml-1-doc/kpml.html.

Dalal, M., S.K. Feiner, K.R. McKeown, D. Jordan, B. Allen, and Y. alSafadi. 1996. Magic: An experimental system for generating multimedia briefings about post-bypass patient status. In *Proceedings of the American Medical Informatics Association*, Washington, D.C., October.

Dale, Robert. 1995. Generating one-anaphoric expressions: Where does the decision lie? In *Working Papers of Pacling-II*, Brisbane, Australia. Available as cmp-lg/9505022.

Elhadad, M. 1990. Types in functional unification grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Detroit, MI. ACL.

Elhadad, M. 1993a. Fuf: The universal unifier - user manual, version 5.2. Technical Report CUCS-038-91, Columbia University.

Elhadad, M. 1993b. *Using argumentation to control lexical choice: a unification-based implementation.* Ph.D. thesis, Computer Science Department, Columbia University.

Elhadad, M. 1995. Using argumentation in text generation. *Journal of Pragmatics.* (To appear).

Elhadad, M. 1996. Lexical choice for complex noun phrases. *Machine Translation.* To appear.

Elhadad, M. and J. Robin. 1992. Controlling content realization with functional unification grammars. In R. Dale, H. Hovy, D. Roesner, and O. Stock, editors, *Aspects of automated natural language generation.* Springer Verlag, pages 89–104.

Elhadad, Michael, Kathleen McKeown, and Jacques Robin. 1997. Floatings constraints in lexical choice. *Computational Linguistics*, June.

Fawcett, R.P. 1987. The semantics of clause and verb for relational processes in english. In M.A.K. Halliday and R.P. Fawcett, editors, *New developments in systemic linguistics.* Frances Pinter, London and New York.

Fries, P. 1970. *Tagmeme sequences in the English noun phrase.* Number 36 in Summer institute of linguistics publications in linguistics and related fields. Benjamin F. Elson for The Church Press Inc., Glendale, CA.

Gross, M. 1984. Lexicon-grammar and the syntactic analysis of french. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 275–282. COLING.

Halliday, M.A.K. 1994. *An introduction to functional grammar.* Edward Arnold, London. 2nd Edition.

Iordanskaja, L., M. Kim, R. Kittredge, B. Lavoie, and A. Polguère. 1994. Generation of extended bilingual statistical reports. In *Proceedings of the 15th International Conference on Computational Linguistics.* COLING.

Kay, M. 1979. Functional grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society.*

Knight, K. and V. Hatzivassiloglou. 1995. Two-levels, many-paths generation. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 252–260, Boston, MA. ACL.

Kukich, K. 1983. *Knowledge-based report generation: a knowledge engineering approach to natural language report generation.* Ph.D. thesis, University of Pittsburgh.

Kukich, K., K. McKeown, J. Shaw, J. Robin, N. Morgan, and J. Phillips. 1994. User-needs analysis and design methodology for an automated document generator. In A. Zampolli, N. Calzolari, and M. Palmer, editors, *Current Issues in Computational Linguistics: In Honour of Don Walker.* Kluwer Academic Press, Boston.

Lester, J.C. 1994a. *Generating natural language explanations from large-scale knowledge bases.* Ph.D. thesis, Computer Science Department, Universtity of Texas at Austin, New York, NY.

Lester, J.C. 1994b. *Generating natural language explanations from large-scale knowledge bases.* Ph.D. thesis, Computer Science Department, Universtity of Texas at Austin.

Levi, J. 1978. *The syntax and semantics of complex nominals.* Academic Press.

Levin, B. 1993. *English verb classes and alternations: a preliminary investigation.* University of Chicago Press.

Mann, W.C. and S. Thompson. 1987. Rhetorical structure theory: description and constructions of text structures. In Gerard Kempen, editor, *Natural Language Generation: New Results in Artificial Intellligence, Psychology and Linguistics.* Martinus Ninjhoff Publishers, pages 85–96.

McKeown, K. and D. Radev. 1995. Generating summaries of multiple news articles. In *Proceedings of SIGIR*, Seattle, Wash., July.

McKeown, K., J. Robin, and M. Tanenblatt. 1993. Tailoring lexical choice to the user's vocabulary in multimedia explanation generation. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics.* ACL.

Mel'cuk, I.A. and N.V. Pertsov. 1987a. *Surface-syntax of English, a formal model in the Meaning-Text Theory.* Benjamins, Amsterdam/Philadelphia.

Mel'cuk, Igor and Nikolaj Pertsov. 1987b. *Surface Syntax of English — A Formal Model within the Meaning-Text Framework.* Linguistic and Literary Studies in Easter Europe. John Benjamins Publishing Co., Amsterdam/Philadelphia.

Meteer, M. 1992. *Expressibility: The problem of efficient text planning.* Pinter, London.

Meteer, M.W., D.D. McDonald, S.D. Anderson, D. Forster, L.S. Gay, A.K. Huettner, and P. Sibun. 1987. Mumble-86: Design and implementation. Technical Report COINS 87-87, University of Massachussets at Amherst, Ahmerst, Ma.

Passoneau, R., K. Kukich, J. Robin, V. Hatzivassiloglou, L. Lefkowitz, and H. Jing. 1996. Generating summaries of workflow diagrams. In *Proceedings of the International Conference on Natural Language Processing and Industrial Applications (NLP-IA'96)*, Moncton, New Brunswick, Canada.

Pollard, Carl and Ivan A. Sag. 1994. *Head Driven Phrase Structure Grammar.* University of Chicago Press, Chicago.

Quirk, R., S. Greenbaum, G. Leech, and J. Svartvik. 1985. *A comprehensive grammar of the English language.* Longman.

Reiter, E.B. 1995. Nlg vs templates. In *Proceedings of the 5th European Workshop on Natural-Language Generation (ENLGW-95)*, Leiden, The Netherlands.

Robin, J. 1994. Revision-based generation of natural language summaries providing historical background: corpus-based analysis, design, implementation and evaluation. Technical Report CU-CS-034-94, Computer Science Department, Columbia Universtity, New York, NY. PhD. Thesis.

Robin, J. and K. McKeown. 1996. Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence*, 85, August. Special Issue on Empirical Methods.

Smadja, F.A. and K. McKeown. 1991. Using collocations for language generation. *Computational Intelligence*, 7(4):229–239, December.

Thompson, S.A. and R.E. Longacre. 1985. Adverbial clauses. In T. Shopen, editor, *Complex constructions*, volume 2 of *Language typology and syntactic description.* Cambridge University Press.

Vendler, Zeno. 1968. *Adjectives and Nominalizations.* Mouton, The Hague, Paris.

Winograd, T. 1983. *Language as a cognitive process.* Addison-Wesley.

Wittenburg, K. 1995. Visual language parsing: if i had a hammer ... In *Proceedings of the International Conference on Cooperative Multimodal Communication*, pages 17–33, Eindhoven, The Netherlands.

Appendix A: Some Generators Using SURGE

The following eight applications served as testbeds for SURGE's ability to be used "as is" for entirely new applications. The practical plug-in usability and versatility of SURGE is demonstrated by the distance between the application domains and research emphasis of these respective systems and the strategy they use to generate the skeletal thematic trees that they pass on to SURGE:

- COMET (McKeown, Robin, and Tanenblatt, 1993), which generates natural language explanations coordinated with graphics in an multi-media tutorial and troubleshooting system for a military radio. It focuses on combining an wide variety of constraints on lexical choice, including the accompanying illustration, the user's vocabulary and previous discourse in addition to the usual syntax and domain semantics.

  It composes a SURGE input by using FUF to unify text plan fragments with a word-based lexicon. It relies on a co-routine control strategy allowing the lexicon to request partial re-planning of textual fragments when it reaches an impasse while combining constraints.

- COOK (Smadja and McKeown, 1991) generates stock market reports from semantic messages summarizing the daily fluctuations of several financial indexes.

  It focuses on the semi-automatic acquisition of a declarative lexicon including collocations and idioms statistically compiled from a large textual corpus. It composes a SURGE input by using FUF to unify this declarative lexicon with the input semantic message. It relies on a fixed, bottom-up control strategy driven by output syntactic function: first choose the verb arguments, then the verb and finally the adjuncts.

- Abella's system (Abella, 1994) generates visual scene descriptions from camera input in two domains: map-aided navigation and kidney X-rays analysis. Its strategy to compose SURGE inputs is entirely geared toward its very specific research focus: validating a fuzzy lexical semantic model of English spatial prepositions. It thus first performs visual reasoning to pick the prepositions best describing the main spatial relations among the input picture landmarks and then procedurally fills locative clause FD templates associated with each chosen preposition.

- SUMMONS (McKeown and Radev, 1995) generates summaries of multiple newswire updates on a given terrorist event from gist story templates extracted from the updates by message understanding systems.

- MAGIC (Dalal et al., 1996) generates multimedia summaries of ICU patient condition coordinating written text, speech and graphics. It focuses on producing summaries tailored to various types of users (*e.g.,* specialist, intern, nurse, administrator) and on the complex temporal and spatial coordination of the various media.

- KNIGHT (Lester, 1994a) generates paragraph-long explanations of scientific processes from a large knowledge base in biology. It focuses on the robustness of the generator when used in a vast domain and the evaluation of the produced output. It composes SURGE inputs procedurally by filling FD templates using information extracted from the knowledge base.

- PLANDOC (Kukich et al., 1994) generates documentation of telephone network extension and upgrade proposals by planning engineers, from the trace of the simulation system that they use to come-up with their proposals.

  Its focuses on high paraphrasing power and on aggregation of related semantic messages into complex clauses. It composes SURGE inputs using a strategy similar to ADVISOR-II, but using constraints derived from the extended discourse instead of argumentative orientation.

- FLOWDOC (Passoneau et al., 1996) generates executive summaries of workflow diagrams acquired and displayed using the SHOWBIZ (Wittenburg, 1995) graphical business re-engineering tool.

  It focuses on pointing out characteristics of the workflow relevant to re-engineering but obscured by the diagram complexity. It composes SURGE inputs using a strategy similar to STREAK's initial draft building stage, except for the combination of compositional generation from a word-based lexicon of general workflow terms together with canned phrases for task-specific operations (entered by the user during workflow acquisition).

- ZEDDOC generates summaries of web page commercial exposure from the access log of the web pages carrying them.

Appendix B: Input Specification Features

The following tables summarize SURGE's input specification language, and illustrate its coverage by example.

Process type specialization				**Example sentence**		
event	material	Agentive non-effective	simple	Michael *Agent*	squats	
			with range	Michael *Agent*	takes	a shower *Range*
		agentive effective	creative	Michael *Agent*	makes	falafels *Created*
			dispositive	Michael *Agent*	eats	falafels *Affected*
		effective non-agentive		Falafels *Affected*	cook	
			creative	Windows *Created*	pop	
	mental			Francisco *Processor*	thinks	
				Francisco *Processor*	liked	how he won *Phenomenon*
	verbal			Michael *Sayer*	speaks	
				Michael *Sayer*	talked	to Cathie *Addressee*
				Michael *Sayer*	said	strange stuff *Verbalization*
relation	ascriptive		attributive	Michael *Carrier*	is	very busy *Attribute*
			equative	The hunter *Identified*	is	the hunted *Identifier*
	possessive			Francisco *Possessor* *Identified*	owns	a big boat *Possessed* *Identifier*
			attributive	Francisco *Possessor* *Carrier*	has	long hair *Possessed* *Attribute*
	locative	natural		It $\emptyset$	rains	
		existential		There $\emptyset$	is	a catch *Located*
		accompaniment		Michael *Located*	is	with his wife *Accompaniment*
		temporal		The end *Located*	is	soon Time
		spatial	attributive	Michael *Located* *Carrier*	is	far away *Location* *Attribute*
			equative	The tree *Located* *Identified*	reaches	the roof *Location* *Identifier*

Table 1
Hierarchy of simple processes

Event Type		**Relation Type**		**Example sentence**			
agentive	non-effective	ascriptive	attributive	Johnson *Agent* *Carrier*	became	rich *Attribute*	
			equative	The Bulls *Agent* *Identified*	became	the Champs *Identifier*	
		possessive	attributive	Orlando *Agent* *Carrier*	picked	Shaquille *Attribute*	
		locative		Seikaly *Agent* *Carrier*	went	to Miami *Located*	
	dispositive	ascriptive		Riley *Agent*	made	New York *Affected* *Carrier*	a good team *Attribute*
				Riley *Agent* *Carrier*	made	New York *Affected*	a good coach *Attribute*
			equative	The Nets *Agent*	made	Coleman *Affected* *Identified*	the richest *Identifier*
		possessive	attributive	The Nets *Agent*	gave	Coleman *Affected* *Possessor*	more money *Possessed*
		locative		Price *Agent*	threw	the ball *Affected* *Located*	out of bounds *Location*
	creative	ascriptive		Peter *Agent*	brews	his beer *Created* *Carrier*	very strong *Attribute*
		locative		Michael *Agent*	opened	windows *Created* *Located*	on the screen *Location*
non-agentive	dispositive	ascriptive		The game *Affected* *Carrier*	turned	wide open *Attribute*	
		possessive		Coleman *Affected* *Carrier*	received	an offer *Attribute*	
		locative		Coleman *Affected* *Located*	fell	on the floor *Location*	
	creative			The windows *Created* *Located*	popped	on the screen *Location*	

Table 2
Hierarchy of composite processes

Mood features			**Example sentence**
finite	declarative		*The Knicks won the game*
	interrogative	yes-no	*Did the Knicks win the game?*
		wh	*Who won?*
		wh-partial	*Whose shirt did they wash?*
	bound	nominal	*That the Knicks won the game* was expected
		adverbial	Ewing scored 28 points *as the Knicks won the game*
	relative	simple	the game *that the Knicks won*
		embedded	the team *against which the Knicks won the game*
non-finite	imperative		*Win that game!*
	participle	present	The Knicks celebrated *after winning the game*
		past	*Once the game won*, the Knicks celebrated
	infinitive	to	The Knicks are able *to win the game*
		bare	Ewing helped the Knicks *win the game*
		for-to	Ewing must dominate *for the Knicks to win the game*
	verbless		*Although without Ewing*, the Knicks won the game

Table 3
Mood hierarchy

Semantic			**Syntactic**	**Example sentence**	
class	**role**	**feature**	**category**		
Locative	Location		Adverb	Bo kissed her **there**.	1
			PP	Bo kissed her **on the cheek**.	2
			finite S	Bo kissed her **where she wanted**.	3
			verbless S	Bo kept the keys **where convenient**.	4
	Direction		Adverb	Bo slid **down**.	5
			NP	Bo drove **this way**.	6
			PP	Bo ran **up the hill**.	7
	Destination		Adverb	Bo went **home**.	8
			PP	Bo sent the letter **to LA**.	9
			finite S	She sent Bo back **where he belongs**.	10
	Path		PP	Bo traveled **via Denver**.	11
	Distance		NP	Bo came **a long way**.	12
Temporal	Duration		Adverb	Bo stayed there **forever**.	13
			NP	Bo stayed there **a long time**.	14
Process	Manner		PP	Bo kiss her **with love**.	15
	Means		Adverb	Bo was treated **surgically**.	16
			PP	Bo was treated **by surgery**.	17
	Instrument	polar+	PP	Bo pushed him **with both hands**.	18
		polar-	PP	Bo negotiated **without an agent**.	19
	Comparison		finite S	Bo went there, **as he did yesterday**.	20
			infinitive S	Bo played hard, **as if to send the fans a message**.	21
			-ing S	Bo played great, **as if peeking on schedule**.	22
			-ed S	Bo played hard, **as if not bothered by his knee**.	23
			verbless S	Bo played great, **as if in great form**.	24
Respect	Matter		PP	Bo talked **about his contract**.	25
Domain	Score		Score	New York beat Indiana **90-87**.	26

Table 4
Predicate Adjuncts

Semantic			Syntactic	Example sentence	
class	**role**	**feature**	**category**		
Locative	Location		Adverb	**There**, Bo kissed her.	1
			PP	**On the platform**, Bo kissed her.	2
	Origin		PP	**From Kansas City**, Bo called Gwen.	3
	Distance		PP	**For a few miles**, the road is damaged.	4
Temporal	Time		Adverb	**Yesterday**, Bo triumphed.	5
			NP	**Last year**, Bo triumphed.	6
			PP	**On Monday**, Bo triumphed.	7
			finite S	**As he received the ball**, Bo smiled.	8
			-ing S	**After receiving the ball**, Bo smiled.	9
			-ed S	**Once injured**, Bo grimaced.	10
			verbless S	**Once on the floor**, Bo grimaced.	11
	Duration		PP	**For a long time**, Bo stayed here.	12
			finite S	**Until Bo recovered**, they waited for him.	13
			-ing S	**Since joining the Raiders**, Bo shines.	14
			-ed S	**Until forcibly removed**, they will stay.	15
			verbless S	**As long as necessary**, they will stay here.	16
	Frequency		Adverb	**Often**, Bo scored twice.	17
			NP	**Twice a week**, Bo runs 10 miles.	18
			PP	**On Sundays**, Bo run 10 miles.	19
Causative	Reason		PP	**Because of injury**, Bo did not play.	20
			finite S	**Because he was injured**, Bo did not play.	21
	Purpose		PP	**For enough money**, Bo would play anywhere.	22
			finite S	**So he would get a raise**, Bo held out.	23
			infinitive S	**To gain free-agency**, Bo held out.	24
	Behalf		PP	**For the Raiders,** Bo scored twice.	24
Determinative	Addition		-ing S	**In addition to trading Bo**, they waived Mike.	25
	Accomp-	polar+	PP	**With her**, Bo would go anywhere.	26
	-animent	polar-		**Without her**, Bo wouldn't go anywhere.	27
	Opposition		PP	**Against the Knicks**, the Nets are 3-1.	28
Process	Manner		Adverb	**Tenderly**, Bo kissed her.	29
	Means		-ing S	**By acquiring Bo**, they strengthen their defense.	30
	Comparison	polar+	PP	**Like Mike**, Bo soared above the rim.	31
		polar-	PP	**Unlike Mike**, Bo can bat.	32

Table 5
Sentence Adjuncts

Semantic			Syntactic	Example sentence	
class	**role**	**feature**	**category**		
Temporal Causative Blend	Co-Event	habit+	finite S	**Whenever you hurt**, call me.	1
		habit-	-ing S	**With his knees hampering him**, Bo's defense is sloppy.	2
			ed S	**Injured against the Giants**, Bo didn't play.	3
			verbless S	**Unable to play due to injury**, Bo stayed home.	4
		habit+	verbless S	**Whenever in doubt**, call me.	5
Causative	Reason		finite S	**Since he was injured**, Bo did not play.	6
	Result		finite S	They wouldn't give him a raise, **so Bo held out**.	7
			infinitive S	They waived Bo, **only to see him flourish elsewhere**.	8
Conditional	Condition	polar+	finite S	**If he is fully fit**, Bo will play.	9
			-ed S	**If well-conditioned**, Bo will play.	10
			verbless S	**If in great shape**, Bo will play.	11
		polar-	finite S	**Unless he is fully fit**, Bo won't play.	12
			-ed S	**Unless well-conditioned**, Bo won't play.	13
			verbless S	**Unless in great shape**, Bo won't play.	14
	Concessive Condition		finite S	**Even if he is fully fit**, Bo won't play.	15
			-ed S	**Even if well-conditioned**, Bo won't play.	16
			verbless S	**Even if in great shape**, Bo won't play.	17
	Concession		PP	**In spite of his injury**, Bo played.	18
			finite S	**Although he was injured**, Bo played.	19
			-ed S	**Although injured**, Bo played.	20
			-ing S	**Although hurting**, Bo played.	21
			verbless S	**Although out of shape**, Bo played.	22
Determinative	Contrast		PP	**As opposed to many others**, Bo never held out.	23
			finite S	**Whereas Mike keeps shooting**, Bo prefers passing.	24
	Exception		PP	**Except Laettner**, all Olympian were pros.	25
			-ing S	**Except for blocking shots**, Bo can do it all.	26
	Inclusion		PP	Bo scored 30 points, **including 5 three-pointers**.	27
			-ing S	Bo can do it all, **including blocking shots**.	28
			verbless S	Bo played everywhere, **including in New York**.	29
	Substitution		PP	**Instead of Mike**, they picked Bo.	30
			-ing S	**Rather than drafting Mike**, they picked Bo.	31
			infinitive S (bare)	**Rather than shoot**, Bo made the perfect pass.	32
	Addition		PP	They picked Bo, **as well as Mike**.	33
Respect	Matter		PP	**Concerning Bo**, they were wrong.	34
	Standard		PP	**For a center**, Bo is very quick.	35
	Perspective		PP	**As a scorer**, Bo remains prolific.	36

Table 6
Disjuncts

Grammatical function	**Syntactic category**	**Example NP**
Pre-Determiner	-	**all of** his first ten points
Determiner	article	**a** victory
	demonstrative Pronoun	**this** victory
	question Pronoun	**what** victory?
	possessive Pronoun	**their** victory
	NP	**New York's** victory
Ordinal	Simple Numeral	the **third** victory
	Numeral Phrase	their **third straight** victory
	Discontinuous Numeral Phrase	their **third** victory **in a row**
Cardinal	simple Numeral	**seven** victories
Quantifier	-	Twice as **many** points
Describer	Adjective	an **easy** victory
	Present Participle	a **smashing** victory
	Past Participle	a **hard fought** victory
Classifier	Noun	a **road** victory
	Adjective	a **presidential** victory
	Present Participle	a **winning** streak
	Noun compound	a **franchise record** victory
	Measure	a **33 point** performance
Head	common Noun	a **victory**
	Proper Noun	the hapless **Denver Nuggets** who lost again
	Noun compound	his league **season high**
	Measure	a career high **33 points**
	Partitive	a season best **12 of 16 shots**
Qualifier	PP	a victory **over the Nets**
	Relative S	a victory **that will be remembered**
	To-infinitive S	a victory **to be remembered**
	For-to-infinitive S	a victory **for them to grab**
	Present-participle S	a victory **invigorating the Knicks**
	Past-participle S	a victory **hard-fought until the end**

Table 7
Syntactic functions inside the NP

Definite	yes	no	
Countable	yes	no	
Partitive	yes	no	
Interrogative	yes	no	
Possessive	yes	no	
Number	singular	dual	plural
Reference-number	singular	dual	plural
Distance	far	near	
Selective	yes	no	
Total	+	-	no
Exact	yes	no	
Orientation	+	-	none
Degree	+	-	none
Evaluative	yes	no	
Evaluation	+	-	
Status	none	same	different...
Case	objective	subjective	genitive
Head-cat	pronoun	common	proper
denotation-class	quantity	season	institution...
Comparative	yes	no	
Superlative	yes	no	
Cardinal	number		
Ordinal	number	last	next (+) or previous (-)
Fraction	numerator	denominator	

Table 8
Features controlling the determiner sequence