Generating Event Descriptions with SAGE: a Simulation and Generation Environment

Marie Meteer
BBN Systems and Technologies
Cambridge, MA 02138
MMETEER@BBN.COM
FAX: (617) 648-1735

ABSTRACT

The SAGE system (Simulation and Generation Environment) was developed to address issues at the interface between conceptual modelling and natural language generation. In this paper, I describe SAGE and its components in the context of event descriptions. I show how kinds of information, such as the Reichenbachian temporal points and event structure, which are usually treated as unified systems, are often best represented at multiple levels in the overall system. SAGE is composed of a knowledge representation language and simulator, which form the underlying model and constitute the "speaker"; a graphics component, which displays the actions of the simulator and provides an anchor for locative and deictic relations; and the generator SPOKESMAN, which produces a textual narration of events.

1. INTRODUCTION

In Text Generation, some of the most interesting issues lie at the interface between the conceptual model (the underlying program) and the generator. It is well recognized that one cannot produce sophisticated text from an impoverished underlying model (McKeown & Swartout 1988). McDonald (1993) makes an even stronger claim:

> Nevertheless, the influence it [the application program] wields in defining the situation and the semantic model from which the generator works is so strong that it must be designed in concert with the generator if high quality results are to be achieved.

In fact, some of the best results in text generation have come from efforts where the model and the generator were developed in tandem, from Davey's early work on describing tic-tac-toe games (Davey 1974) to Dale's recent work on generating recipes (Dale 1990). Dale found that in order to generate referring expressions in recipes, he had to work on the representation of the underlying objects and their state changes in order to be able to correctly generate the number of the noun phrases in examples such as "Grate one carrot...Add the carrots to the stew". The most impressive results to date in event generation is the NAOS system (Novak 1987, Neumann 1989), which produces natural language descriptions of object movements in a street scene. It is designed to take is input from a vision system observing traffic, which captures both temporal and spatial relationships among the objects in the scene. The focus of the work has been on representing events and the relations among them and then connecting those events to case frames for expressing them in natural language.

In narration, temporal and aspectual information must be available in the underlying model in order to describe events. For example, using the well recognized Reichenbachian model, three different temporal points, point of event (E), point of speech (S), and point of reference (R), are needed in order to adequately account for the English tense system, as shown in the following examples:

1. *Peter drove to work.* ($E = R < S$)
2. *Peter had driven to work.* ($E < R < S$)

Such problems are generally treated as unified systems in linguistics within studies of semantics or the lexicon. However, in generation research, the issue is not just what distinctions there are, but at what level (model, text planner, syntactic component) should the information needed to make these distinctions be represented. Taking the temporal points in the Reichenbachian model as an example, two of the points, E and S, are facts of the model, when the event took place and the time the speaker is producing the utterance. However, the third point, the reference time, is a fact of the discourse, a choice to be made by the speaker. (1) and (2) above are distinguished by the reference time, but otherwise could describe the same event and be spoken at the same time.

While most studies of events are done within the realm of linguistics, where the focus in on the expression of event descriptions, it is clear that the way events are modelled is also an essential element. Bach (1988) describes "how certain metaphysical assumptions are essential to an understanding of English tenses and aspects. These assumptions have to do with the way reality—or our experience—is structured."

From a generation perspective, there are two basic questions to be answered. First, what information is needed in order to produce the distinctions available in language, and secondly, what distinctions are facts of language (and thus should be in the generator) and which are better represented at the model level?

The problem of finding a general way to research such questions has led to the development of SAGE, a "Simulation and Generation Environment", which provides components for both conceptual modelling and text production. In SAGE, a frame-based knowledge representation component models objects and their properties, an event-based simulator models the actions of multiple agents, and a graphics component provides models of the physical geography in the virtual world, in addition to providing a visual interface to the objects, agents, and actions. Text generation is provided by the SPOKESMAN system. SPOKESMAN is data directed in that it links to the other components both through mappings from concepts in the knowledge representation and through instances of objects and events created by the simulator.

In this paper, I describe the components of SAGE and how they are integrated, focusing on the generation of event descriptions. In Section Two, I look at what information is needed to generate events through analysis of events and a review of the linguistic literature. In Section Three, I describe the architecture of SAGE and its representational levels, including where in the overall system event information is represented and in Section Four I illustrate these issues using paragraphs generated in SAGE, such as the following:

Fluffy wants to catch a mouse. He is looking for her.
The mouse wants to get cheese. She is leaving a mouse-house. She is going toward it.
Fluffy is chasing the mouse. He is going toward her. He caught her.
The mouse didn't get the cheese.

The overall methodology applied in this research tooapproach the problem from two directions, as depicted in Figure 1. One direction is that from a situation modelled in some application program to the expression of some set of goals from that program in a natural language (in this case, English). The second direction is the use of text analysis to work backwards from the way something is said to what decision points led to that text, what alternative choices were not made, which decisions were constrained by the syntax or lexicon of the language, and what information is needed in the application program in order to make these decisions.

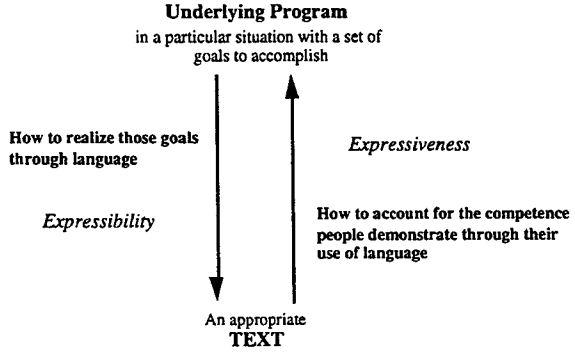

Figure 1: Bi-directional approach to generation research

This methodology is exemplified in the work presented here in the first direction by the use of SAGE to model situations and generate text (described in Section 3 and exemplified in Section 4) and in the second direction, through the analysis of events and projection of that analysis onto the decisions of the generator (described in Section 2).

2. EVENTS

In this section, we address the problem of representing and describing events. The goal is to identify the information that needs to be represented in order to take advantage of all the resources a language provides for describing events (which involves determining which distinctions language supports) and determining at what level the information should be represented and the decisions made to make those distinctions.

We first outline six different kinds of information needed for the expression of events: linear time, event type, temporal modifiers, event structure, argument structure, and agency. In section three, we describe the architecture of SAGE and show where the decisions supporting the distinctions in the expression of events are made.

2.1 Information for Events

First, in order to generate events, there needs to be a model of linear time. Most of the current work on tenses is based on a Reichenbachian-style analysis, which involves three temporal notions: point of speech, point of the event, and point of reference, as we showed above in examples (1) and (2).

Another well recognized distinction is that of event types, such as state, process, transition, exemplified by the following examples:

3. *The mouse is under the table. (state)*
4. *Fluffy ran. (process)*
5. *Peter found his keys. (transition—achievement)*
6. *Helga wrote a letter. (transition—accomplishment)*

While verbs have an intrinsic type (e.g. wait is a process and catch is a transition), these types also apply to whole phrases, since tense, aspect, adjuncts and arguments can compose with the type of the lexical head to form a new type:

7. *Fluffy ran into the kitchen. (process —> transition)*
8. *Helga is writing a letter. (transition —> process)*
9. *The mouse is caught. (transition —> state)*
10. *Roscoe builds houses. (transition —> iteration)*

Four kinds of temporal adverbials can be distinguished and are linked to the event types. *Duration* modifies processes, as in example (11a), but not transitions (11b); *frame* adverbials modify accomplishments, as in (12a), but not processes (12b); *point* adverbials modify achievements, as in (13); and *frequency* adverbials modify iterative events, as in (14).

11. a) Peter waited in the lobby for an hour.
*b) * Helga wrote the letter for an hour.*
12 a) Helga wrote the letter in an hour.
*b) * Peter waited in the lobby in an hour.*
13. Hank found the pen at four o'clock.
14. Martha writes letters frequently.

It is also clear that events are not undifferentiated masses, but rather have subparts that can be picked out by the choice of phrase type or the addition of adverbial phrases. Moens & Steedman (1988) identify three constituents to an event nucleus, a preparatory process, culmination, and consequent state, whereas Nakhimovsky (1988) identifies five: preparatory, initial, body, final, result, exemplified by the following:[1]

15. When the children crossed the road,
a) they waited for the teacher to give a signal.
b) they stepped onto its concrete surface as if it were about to swallow them up.
c) they were nearly hit by a car.
d) they reached the other side stricken with fear.
e) they found themselves surrounded by strangers.

Pustejovsky (1991) offers a much more compositional notion of event structure, where a transition is the composition of a process and a state. This analysis is more closely tied to the lexicon than Moens and Steedman's or Nakhimovsky's (and is offered in the context of a generative theory of lexical semantics). It not only accounts for the semantics of verbs, but also their compositions with adjuncts to form new types, as in (7) above.

The participants of an event are those entities that act in or are acted upon in the event. The argument structure is the set of participants in the event that are grammaticized with respect to a particular lexicalization of the event, such as the agent, theme, source, and goal. For some event types (especially those that appear as examples in linguistics papers), the distinction between what is an argument and what is an adjunct is clear. For example, in "Fluffy ate a bone in the dining room yesterday", "Fluffy" (the agent) and "a bone" (the theme) are arguments, whereas the location and time are adjuncts. For other verbs, however, the distinction is not so clear, as in "Mickey slid into home plate", where the location is a necessary participant to the meaning, yet as a location it would be treated as an adjunct in most analyses.

Agency in an event is an aspect of the argument structure, but since there are some important generalizations over this participant that is not true of others, we treat it separately. One of the most widely discussed syntactic variations is the active/passive, which vary on the placement/inclusion of the agent. As discussed in Meteer (1991) there are really many different motivations for what is often characterized as a single "switch" in generators. The degree of explicitness of the agent in different syntactic constructions can be seen in the following set of examples, from the explicit inclusion of the agent in the subject position in (a), to the movement of the agent to the by-phrase in (b), to the deletion of the agent in (c), to an adjectival construction in (d) using the past participle form of the verb, to a result construction in (e) that includes no indication of agency. Notice that the explicitness of the event's tense diminishes along with the agency.

18. a) Peter tore the shirt.
b) The shirt was torn by Peter.
c) The shirt was torn yesterday.
d) Peter wore the torn shirt yesterday.
e) No one noticed the tear in the shirt. (cf. No one noticed the missing button.)

Another argument that agency should be treated specially is made by Pustejovsky (1991) in his work in generative lexical semantics and event structure. Pustejovsky argues that some distinctions usually characterized by event type or argument structure are actually rooted in agency, such as the difference between verbs that are lexically transitions but have unaccusative and causative variants ("The door closed" vs. "Thelma closed the door"). Furthermore, the difference between the two types of transitions, accomplishments vs. achievements, is based on an agentive/non-agentive distinction. According to Pustejovsky, accomplishments (such as build, draw, and leave) include both the act and the causation in their semantics, whereas in accomplishments (such as win, find, and arrive) agency is not an intrinsic part of the semantics of the verb, but is rather based on something else, such as the configuration of elements (someone wins when they are at the front in some competition at a particular moment, given some particular evaluation function). This is substantiated by the interaction with "deliberately" and these verbs, shown in the examples below:

19. a. Helga deliberately drew a picture
*b. *Helga deliberately found the pen.*
20. a. Peter deliberately left the party.
*b. *Peter deliberately arrived at the party.*

Having identified the information necessary for the description of events, the next step in the research is to determine which levels should be responsible for the representation of the information. In particular, what aspects of the event description are

- dependent on the event itself (a fact of the world/model);
- dependent on the discourse context;
- dependent on what linguistic resources are available (e.g. lexicon and syntax) and constraints on their composition.

SAGE allows us to approach these questions experimentally, using SAGE to provide a context in which to make the decision about where the information is best represented and the decisions best made. In the next section, I describe SAGE, its components, and how they interact. I also include where in that architecture the information for event descriptions is represented. In

[1] Nakhimovsky, 1988, p.31.

Section Four I look at these issues more concretely using an example narration from SAGE.

3. THE COMPONENTS OF SAGE

SAGE is a package of integrated tools to aid in exploring the relationship between simulated events in a multi-agent environment, the narration of those events by a text generator, and the animation of the events with simple graphics. There are three main components to SAGE:

- The speaker's intensional world is modelled in an "underlying program" built using the knowledge representation language VSFL and the event based simulator SCORE;[2]
- The text generator is SPOKESMAN, with the linguistic component MUMBLE-86 and the text planner Ravel;
- The graphics component is built with the graphics package in Macintosh Common Lisp and Mac Quickdraw.

3.1 The Modelling Component of SAGE

The underlying program of SAGE, that is, the part in which objects and events are modelled, is a knowledge based simulation system with two parts: the knowledge representation language and the simulator. The objects and events are modelled primarily in VSFL (the Very Simple Frame Language), which is an amalgamation of a knowledge representation language and an object oriented programming language. As a descendent of KL-ONE (Brachman & Schmolze 1985), it provides concept and role hierarchies and multiple inheritance of roles (including role restrictions and defaults)[3].

The knowledge base in SAGE is what ties together the main components. It acts as a central resource, providing definitional information for types and relations. The type of an object controls its actions in the simulation, the way it is expressed by the generator, and how it is displayed by the graphics component. For example, if the generator is referring to the object #<fluffy>, which is of type dog, it uses the mapping of concept dog to the class of alternative expressions for named individual (such as using the name, a pronoun, "I" if fluffy is the speaker, a generic reference "a dog" if he is being introduced and not known, etc.). The graphics component uses the fact that the type "dog" inherits from "agent" and agents are drawn using triangles pointing in the direction the agent is facing. There is a core knowledge base which contains the set of concepts that are used by all domains, such as ACTION, OBJECT, LOCATION. This is similar to the upper model used in Penman (Bateman 1989).[4]

Events are represented as goals and procedures in the simulator and are also linked to the knowledge base through their types, which are concepts in the knowledge base. This provides a classification of events into the three main event types: state, process, and transition. The parameters to those goals/procedures are the roles on the concept, defining the participants in the event, as well as associated information, such as location.

The simulator SCORE is an event-based simulator that supports multiple agents executing parallel goals and actions. SCORE provides a language for declaratively representing the plans of agents, where a *plan* is a partial ordering of procedures and subgoals for accomplishing goals and handling contingencies. *Goals* define the intentions of agents (goals succeed or fail) and *procedures* define a sequence of actions and decision points (procedures complete or are interrupted). The primitives in this system are *actions*, which are simply lisp functions.

The hierarchical structure of the plans, with procedures defined in terms of subprocedures and actions, defines the structure of events, in the sense of Nakhimovsky, described above. The procedure for cross-the-road, for example, would be defined in terms of prepare-to-cross (look both ways, wait for traffic, wait for teacher's signal, etc,) step onto the road, walk across, step on to the other side, with a consequent change in that agent's state (more specifically, his location) from one side of the street to another. Note that in these terms, the constituents of an event is a fact of the model and the level of granularity that is represented, and not a linguistic issue. We can describe the event as a single action "cross the road", but with an animation component, each of the steps must be modelled as well (depending, of course, on the granularity of the animation, since if the "road" is a single line, then a single action might be adequate to move the agent across it).

When a goal/procedure is run, an instance of the event concept is created and the parameters are filled with instances of objects and other events. The start and end time and instances of subprocedures are filled in as the procedure runs, providing the event time necessary for the generation of tense. The simulator passes instances of actions to both the generator and graphics component,

---

[2] VSFL ("Very Simple Frame Language") and SCORE ("Sproket Core") were developed at BBN Systems and Technologies by Glenn Abrett and Jeff Palmucci, with assistance from Mark Burstien, and Stephen Deutsch. VSFL is a reimplementation of SFL, which is a descendent of KL-One. SCORE is a reimplementation of the SPROKET simulator. See Abrett, et al. 1989 for a more detailed description of these systems.

[3] VSFL is "very simple" in that it does not support automatic classification and does not have a graphical editor (though it does have a graphical viewer). Its integration with CLOS (Common Lisp Object System) supports the creation of instances and the ability to associate methods with concepts. The integration with CLOS also provides more efficient slot accessors and other optimizations.

[4] As yet we make not theoretical claim to the significance of our choice of which concepts live in the core. This is part of our ongoing research.

which use the type hierarchy to know how to describe the action or how to update the display.

3.2 The Generation Component of SAGE

SPOKESMAN is composed of two major components: the *text planner* and the *linguistic realization component.* The text planner selects the information to be communicated explicitly, determines what perspectives the information should be given (e.g. whether something is viewed as an event, "Peter waited for a long time", or as an object, "the long wait"), determines the organization of the information, and chooses a mapping for the information onto the linguistic resources that the language provides. The linguistic realization component is MUMBLE-86 (McDonald 1984; Meteer, et al. 1987). It carries out the planner's specifications to produce an actual text. It ensures that the text is grammatical and handles all of the syntactic and morphological decision making.

Both components use multiple levels of representation, beginning with objects from the application program through progressively more linguistic representations to the final text, as shown in Figure 4.

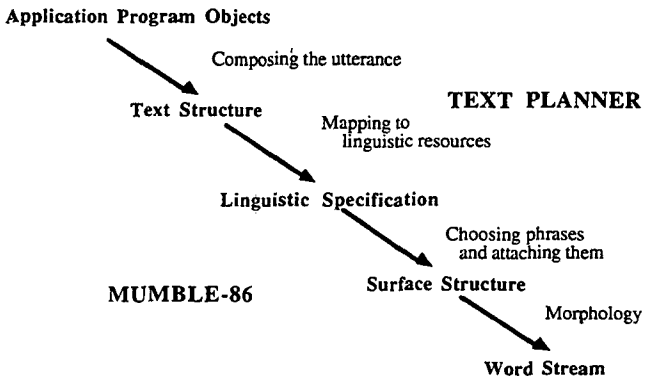

Figure 4: SPOKESMAN

Each representational level is a complete description of the utterance and provides constraints and context for the further decisions that go into its construction. This is an essential part of planning by progressive refinement, because the representation must constrain the planner so that it is not allowed to make decisions it will later have to retract. The representational levels also control the order of the decisions.

The Text Structure, which is the central representation level of the text planner, provides a vocabulary for mapping from the terms of the model level to the linguistic terms of the generator. It is at this level that the content lexical items are selected and the semantic category of the constituents is determined. Events and their composition are handled in the style of Pustejovsky (described above). For example, a RUN-TO-LOCATION procedure in the simulator (which has a type of transition) is mapped to the composition of the lexical head "run" (with the agent from the WHO role of the procedure), which is lexically a process, with a goal locative adjunct (e.g. "to the kitchen"), which produces a transition as shown in the Text Structure tree in Figure 5. Constraints on the transition type indicate that only a frame adverbial (e.g. "in two minutes"), can be added, and not a duration (e.g. "for two minutes").

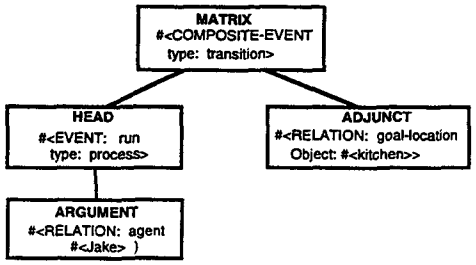

Figure 5: Text Structure Tree

The speaker could also choose not to express the entire transition as a kernel unit, but rather pick out only the process portion, as in "Jake ran", in which case the composition would also be of type process, which constrains the temporal adjuncts to be of type duration, rather than frame. (See Meteer, 1992, for a more complete description of the vocabulary of terms in the text structure and its role in the incremental composition of the text plan.)

Another role of the Text Structure is to keep track of discourse level information, such as focus and what entities have been referenced and in what context. As Webber (1988) points out, tense can be used anaphorically, just as definite nps and pronouns can, and the speaker must keep track of the changing temporal focus. It is the combination of the discourse specific information and the event time and speech time as defined by the simulator[5] that are needed to correctly generate English tense, as described above.

4. EXAMPLE

In this section, we look at the underlying structures for a narration of a simulation in the SAGE system. We focus on those elements at the interface between the underlying program and the generator. The simulation begins with each of the agents located at a position on the map (Figure 6). Fluffy the dog is assigned a goal of catching a mouse and Jake the mouse is assigned the goal of getting some cheese, which is located in the kitchen. The following simple paragraph, generated by Spokesman, describes each of their goals and actions and is produced incrementally as they are executed by the simulator:

[5] The simulator is the "speaker" in SAGE, since it is the component that has goals to express information and the model defined by the knowledge base is the intensional model of that speaker. The generator defines the possibilities for expression and executes the speaker's goals.

Fluffy wants to catch a mouse. He is looking for her.
The mouse wants to get cheese. She is leaving a mouse-house. She is going toward it.
Fluffy is chasing the mouse. He is going toward her. He caught her.
The mouse didn't get the cheese.

Example paragraph

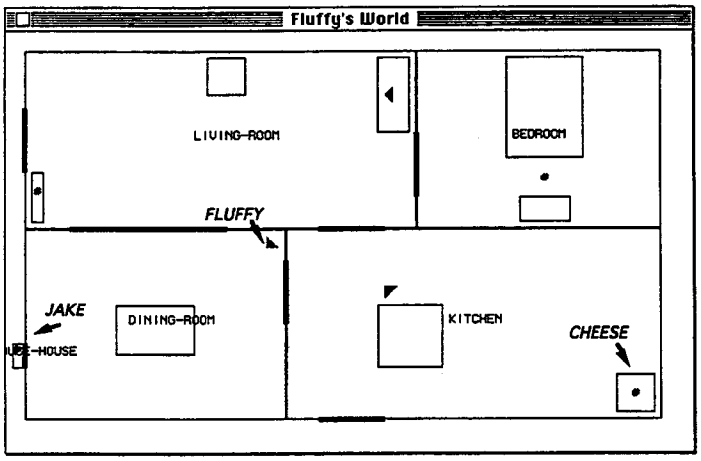

Figure 6: Map of Fluffy's house

As described in Section 2 above, there are several different kinds of information needed to generate event descriptions. Since the underlying program in this case is an ongoing simulation, the linear time is easily available in the system. Figure 7 shows a graph of the events as they are created in the system, marked by their time. Since the generation is a "play-by-play" narration, the event time, reference time, and speaker time are usually the same, as is reflected in the use of the present tense in the text. An exception to this can be seen at the end of the above paragraph. Since the actions underlying these sentences are marked as completed by the simulator, the event time is before the speaker time, and thus the past tense is used.

Another kind of information needed for generation is the event type. Note that in SAGE there is not a single notion of "event type", but rather two: one for the underlying knowledge base and the other in Ravel, the text planner. This reflects the difference between:

- a concept's intrinsic type in the domain, which includes what objects it is related to (e.g. its parents, what slots it has), and how it functions in the underlying program (e.g. what methods it has or inherits), and
- a concept's "expression type" in the text planner, which reflects the fact that the speaker can alter an object's expression type through lexical choice (e.g. nominalization) and the choice of tense, aspect and adjuncts.

Portions of these two types of classification hierarchies are shown in Figure 8. They are mediated by the mappings in Ravel, which we describe below.Another kind of information that is represented in the underlying program and used by the generator is the difference between a goal, which represents an agent's intentions, and a procedure, which represents an agent's actions. In the example paragraph, this is reflected by the use of the matrix verb "want to" in the first and third sentences in the case where the "action" field of the goal event is "start", and by the use of the past tense in the sentence "He caught her", when the action field is "succeed" and by the past and negation in the sentence "The mouse didn't get the cheese" when the action field is "fail". Instances of goals and procedures are shown in Figure 9. Each simulation event object has two parts: (1) the goal or event wrapper, which indicates the goal/procedure status, the relationship of this event to other events (is a super or sub event), and the time stamp; and (2) the action instance, which is an instance of an action type from the domain model with the fields filled in, indicating the various actors and objects acted on and other related information (note that this information is often but not always expressed as verb arguments).

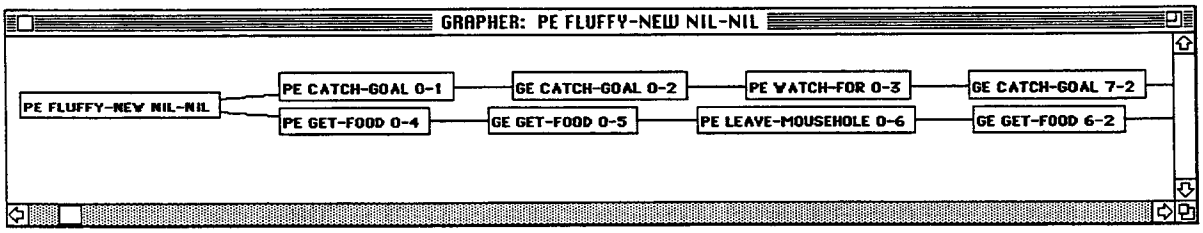

Figure 7: Graph of events in executed simulation

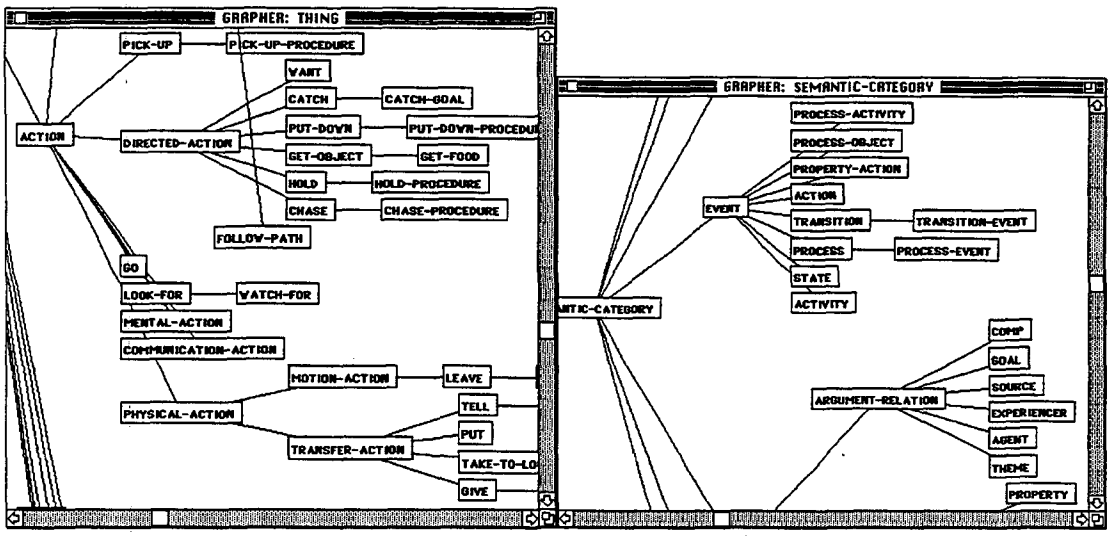

Figure 8: Event types in the Domain Model and Text Planner

```
#<GOAL-EVENT #x3B89F9>
Class: #<STANDARD-CLASS GOAL-EVENT>
Instance slots
ACTION:  START
ACTION-INSTANCE: #<CATCH-GOAL #x3B8699>
SUBS: (#<PROCEDURE-EVENT #x3B9CB1>
       #<PROCEDURE-EVENT #x3BD431>)
SUPER: (#<PROCEDURE-EVENT #x3B8741>)
START-TIME: 0
SUB-TICK-START: 5

#<CATCH-GOAL #x3B8699>
Class: #<STANDARD-CLASS CATCH-GOAL>
Instance slots
WHAT: #<Agent: JAKE>
WHO: #<Agent: FLUFFY>>

#<PROCEDURE-EVENT #x3B9CB1>
Class: #<STANDARD-CLASS PROCEDURE-EVENT>
Instance slots
PROCEDURE-SUB-EVENTS:
        (#<SPROKET-EVENT #x3BA509>
        #<SPROKET-EVENT #x3BD049>)
END-TIME: 1
ACTION-INSTANCE: #<WATCH-FOR #x3B9BD9>
SUPER: (#<GOAL-EVENT #x3B89F9>)
START-TIME: 0
SUB-TICK-START: 6

#<WATCH-FOR #x3B9BD9>
Class: #<STANDARD-CLASS WATCH-FOR>
Instance slots
WHAT: #<Agent: JAKE>
WHO: #<Agent: FLUFFY>
```

Figure 9: Instances of goals and procedures in the simulator

The connection from the underlying program to the text generator is made through the mapping tables. Mapping tables provide an association between a concept in the domain hierarchy and the set of linguistic resources that can be used to express instances of that concept. For example, the mapping tables shown in Figure 10 connect the goal and procedure events shown above to choices in the generator. Note that the mapping is conditional, so the goal event is mapped to a set of alternatives for expressing a state with an activity argument at the level of the Text Structure and to a tree family with the verb "want" when the action field is "start" and just uses the mapping for the action instance in other cases. The procedure event also adds nothing to the mapping, but just uses the mapping for the instance class ("watch-for" in the example above).

```
(mapping-tables  (find-class 'spr::goal-event)
   class-to-text-structure
   ((:condition (eq (spr::action self) 'spr::start)
     :realization-class state-to-activity-class
     :arguments (:agent (spr::who (core-event-object self))
                        :event self
                        :theme (core-event-object self)
                        :time (determine-tense self)))
    (:condition (default)
     :mapping-function remap-with-same-self
     :arguments ((core-event-object self))))

   object-to-tree-family
   (:argument-structure-class state-with-propositional-
complement
    :arguments (`(mumble::verb "want" ))))

(mapping-tables  (find-class 'spr::procedure-event)
   class-to-text-structure
    (:mapping-function remap-with-same-self
     :arguments ((core-event-object self))))
```

Figure 10: Mapping tables for goals and procedures in the text planner

Mapping tables for the action types catch and look-for[6] are shown below in Figure 11. Each has two mappings, one which offers alternatives at the Text Structure level and

[6] I realize there is a confusion here between "look-for" and "watch-for". "Watch-for" is a child of "look-for" in the hierarchy (see Figure 8), and was probably introduced automatically by the system as the name of a procedure of type "look-for". While confusing, this exemplifies the kind of naming problems that come up in real systems, and since all of these examples are directly from running code, I hesitate to white them out. In fact, it is the relations among the concepts and their fields that distinguish them, not their symbol names, and it is the mappings that determine what lexical items are used to express them (though some mappings use the concept name as a default lexical item when none is specified.)

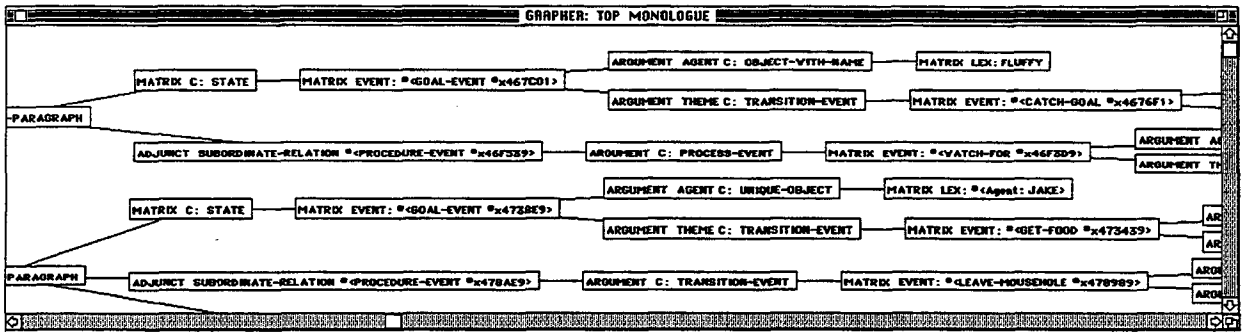

Figure 12: Text Structure

a second which offers choices at the linguistic specification level. Specifically, each realization class that is mapped to at the CLASS-TO-TEXT-STRUCTURE-MAPPING offers alternatives of different semantic expression categories (for example expressing the transition "catch" as a process by using the progressive aspect) and the opportunity to leave out optional arguments (even though they are available in the underlying structure, the speaker can choose to leave them out). The argument structure class inspects the choices that have been made in semantic category and arguments and selects the appropriate tree family. The specific elementary tree will not be selected until the level of the surface structure in Mumble-86, when syntactic context is available.

```
(mapping-tables  (spr::concept 'spr::catch)
 class-to-text-structure
   (:realization-class transition-event-class
    :arguments (:agent (spr::who (core-event-object self))
                :event (core-event-object self)
                :theme (spr::what (core-event-object self))
                :time (determine-tense self)))
  object-to-tree-family
     (:argument-structure-class transitive-event
      :arguments (`(mumble::verb "catch" )) ))

(mapping-tables  (spr::concept 'spr::look-for)
 class-to-text-structure
   (:realization-class process-event-class
    :arguments (:agent (spr::who (core-event-object self))
                :event (core-event-object self)
                :theme (spr::what (core-event-object self))
                :time (determine-tense self)))

  object-to-tree-family
     (:argument-structure-class transitive-prepcomp
      :arguments (`(mumble::verb "look" )
                  `(mumble::prep "for" ))))
```

Figure 11: Mapping tables CATCH

The choices described above result in the Text Structure representation, as shown in Figure 12:

5. CONCLUSION

We have seen that what are generally treated as a single phenomenon stretch across multiple levels in SAGE:

- Event time and speech time are facts of the underlying program, whereas reference time is part of the discourse model in the generator.
- Events have an intrinsic type in the model, but the speaker can make explicit only a portion of the event or compose it with other information and express it as a different event type. What subconstituents of an event are available to be made explicit are defined by the procedures of the underlying program (in this case, the simulator), but the ways they can be made explicit are constrained by the resources of language.
- Whether an action is caused by an agent is part of the definition of the action, but whether that agent is expressed is a choice by the speaker.

In all of these cases, the information must be represented at both the model level and in the generator in order to capture the full expressiveness of event descriptions in English. Using SAGE as an environment in which to model both conceptual and linguistic information lets us experiment with the best division of the information across its components.

References

Abrett, G., Deutsch, S. and Downes-Martin, S. (1989), "AI Languages for Simulation", BBN Technical Report, BBN Systems and Technologies, Cambridge, Massachusetts.

Abrett, G., Burstein, M., & Deutsch, S. (1989), "Tarl: Tactical Action Representation Language, an environment for building goal directed knowledge based simulation", BBN Technical Report No. 7062, BBN Systems and Technologies, Cambridge, Massachusetts.

Bach, Emmon (1981) "On Time, Tense, and Aspect: An Essay in English Metaphysics" in , Academic Press.

Bateman, J., Kasper, R., Moore, J., & Whitney, R. (1989) "A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model" USC/Information Sciences Institute Technical Report.

Brachman, Ronald, & James Schmolze (1985) An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9:197-216, 1985.

Dale, Robert (1990) "Generating Recipes: An Overview of Epicure" in Dale, Mellish, & Zock (eds) Current Research in Natural Language Generation, Academic Press, London, p.229-255.

Davey, A. (1974), *Discourse Production*, Edinburgh University Press. Edinburgh.

McDonald, David D. (1984) Description Directed Control. *Computers and Mathematics 9(1)* Reprinted in Grosz, et al. (eds.) *Readings in Natural Language Processing*, Morgan Kaufman Publishers, California, 1986, pp.519-538.

McDonald, David D. (1993) "Natural Language Generation" to appear in the *Encyclopedia of Language and Linguistic,* Computational Linguistics Section, C. Mellish (ed), Pergamon Press.

McDonald, David D. (1991) "On the Place of Words in the Generation Process" in *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Eds. Cecile Paris, William Swartout and William Mann, p.229-247.

McKeown & Swartout (1988) "Language Generation and Explanation" in Zock & Sabeh (eds.), *Advances in Natural Language Generation*, Pinter Publishers, London, p.1-51.

Meteer, Marie W. (1992) *Expressibility and the Problem of Efficient Text Planning*. Pinter Publishers, London, England.

Meteer, Marie W. (1992) "Portable Natural Language Generation using SPOKESMAN" *Proceedings of the 3rd Conference on Applications in Natural Language Processing*, Trento, Italy, April, 4-6.

Meteer, Marie W. (1991) "SPOKESMAN: Data Driven, Object Oriented Natural Language Generation", *Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications*, Miami Beach, Florida, February 26-28.

Meteer, Marie W. (1991) "Decision Making in Generation: A Multi-leveled Approach", IJCAI-91 Workshop on Decision Making throughout the Generation Process, Sydney, Australia, August, 1991.

Meteer, Marie W., David D. McDonald, Scott Anderson, David Forster, Linda Gay, Alison Huettner, Penelope Sibun (1987) *Mumble-86: Design and Implementation*, UMass Technical Report 87-87, 173 pgs.

Moens, Marc & Steedman, Mark (1988) "Temporal Ontology and Temporal Reference" *Computational Linguistics*, Vol.14, No. 2, p.15-28.

Nakhimovsky, Alexander (1988) "Aspect, Aspectual Class, and the Temporal Structure of Narrative" *Computational Linguistics*, Vol.14, No. 2, p.29-43.

Neumann, Bernd (1989) "Natural Language Description of TIme-Varying Scenes" in *Semantic Structures*, D. Waltz (Ed.) Laurence Erlbaum Associates, New Jersey. p. 167-206.

Novak, Hans-Joachim (1987) "Strategies for generating coherent descriptions of object movements in street scenes" in *Natural Langauage Generation*, G. Kempen (Ed.) Marinus Nijoff Press. p. 117-132.

Pustejovsky, James (1992) "The Syntax of Event Structure" *Cognition*, Vol. 41, 47-81.

Webber, Bonnie (1988) "Tense as Discourse Anaphor" *Computational Linguistics*, Vol.14, No. 2, p.61-73.