Behavior-based Language Generation for Believable Agents

A. Bryan Loyall   Joseph Bates

March 1995

CMU-CS-95-139

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This work was supported in part by Fujitsu Laboratories and Mitsubishi Electric Research Laboratories.

Keywords: artificial intelligence, believable agents, active language, embodied language and action, situated natural language generation, art, entertainment

Abstract

We are studying how to create believable agents that perform actions and use natural language in interactive, animated, real-time worlds. We have extended Hap, our behavior-based architecture for believable non-linguistic agents, to better support natural language text generation. These extensions allow us to tightly integrate generation with other aspects of the agent, including action, perception, inference and emotion. We describe our approach, and show how it leads to agents with properties we believe important for believability, such as: using language and action together to accomplish communication goals; using perception to help make linguistic choices; varying generated text according to emotional state; and issuing the text in real-time with pauses, restarts and other breakdowns visible. Besides being useful in constructing believable agents, we feel these extensions may interest researchers seeking to generate language in other action architectures.

1 Introduction

We are studying how to create believable agents that perform actions and use natural language in interactive, animated, real-time worlds. Such worlds might be built as entertainment, as art, or as interfaces to databases, libraries, or the Internet.

"Believable" is used here in the sense of believable characters in the arts, meaning that a viewer or user can suspend their disbelief and feel that the character or agent is real. This does not mean that the agent must be realistic. In fact, the best path to believability almost always involves careful, artistically inspired abstraction, retaining only those aspects of the agent that are essential to express its personality and its role in the work of which it is part.[1]

While full realism is rarely appropriate, we have found, as have others before us, that fine details can have a great influence on whether a creature seems alive. The use of the eyes, the timing of pauses in speech, an awareness of body position and personal space, are each examples of these important details.

To further bring out some of the requirements on believable language and action producing agents, let us examine four seconds from the film Casablanca [5], which we have transcribed in Figure 1. In this scene, Ugarti (Peter Lorre), a dealer in the black market, is being arrested for stealing two letters of transit. Just before the police haul him away, he seeks help from Rick (Humphrey Bogart), the seemingly cynical owner of the Café Américain. Speech and action occur simultaneously, and we have transcribed them into two columns to show the parallelism.

In these moments, Ugarti is a very believable and engaging character. If we wish to build autonomous agents that can produce similarly believable behavior, we might identify the following as a few of the challenges:

• In general, production (and understanding) of language and action appear very tightly integrated. We feel this probably is not the result of distinct sensing, acting, understanding, and generating modules communicating through narrow channels.

• Action and language are used together to accomplish communication goals. An example is pleading language with a wide eyed facial expression.

• Language generation occurs in parallel with other independent goals. Parallel behaviors producing streams of control signals to multiple channels (eyes, body, voice) help bring the character to life. Ugarti is generating language while watching and struggling with the police.

• Perception and action occur as subgoals of generation. For instance, as the transcript begins, Ugarti yells "Rick" because he perceives that Rick is not attending to him. He acts by putting his hands on Rick's arms to signify an embrace of friends, presumably to increase the persuasiveness of his words. We believe both of these arise most naturally as consequences of generation.

• Generation does not reduce the agent's responsiveness to events in the world. Ugarti notices and responds to the police approaching, grabbing him, etc. all while

[1] There are many discussions of this idea in the arts. Such a discussion together with a wealth of other material particularly useful for animation is presented by Thomas and Johnston [16].

Speech	Action
... Ugarti enters yelling "Rick! Rick! Help me!", puts his hands on Rick's forearms. Rick pushes Ugarti against a column saying "Don't be a fool, you can't get away."	
But Rick, hide me!	U's eyes are wide, focused on R, U has facial expression of extreme desperation and fear.
Do	U's eyes and then head turn left to see approaching police, mouth tight, face tense.
something,	Head, eyes back on R, intense gaze, "something" emphasized.
you	Eyes then head turn a bit left toward police as they grab him.
must	U's face compresses in pain.
help	Shrinks down, looks further away from R.
me	Twists to get free.
Rick!	Looks back at R, but eyes pressed shut, looks away as police pull at him.
Do something!	U looks toward R as he speaks, then away in pain as he is dragged from scene yelling.

Figure 1: Transcript of moment from Casablanca.

producing one short sentence.

• Pauses, restarts, and other breakdowns are desirable when they reflect the personality and situation of the agent. In the fine scale timing of the transcript, the actions of the police absorb some of Ugarti's attention and noticeably vary his rate of speech production.

• Generation is incremental. Word choice and other generation activities seem to be as influenced by the real-time flow of events as other action production.

• Language generation, like other action, varies with personality and emotional state. Ugarti pleading with Rick is in accord with his personality, and with his emotions upon being arrested.

• Emotion is produced from the success and failure of communication as well as of other action. For instance, Ugarti is upset about not escaping the police, but this failure is partly a consequence of not having enough time to convince Rick to help him. So he is angry and sad about his inability to achieve a communication goal, as that was his means to achieve an important parent goal. Anger in response to being constantly interrupted is another example of this phenomenon.

Artists know these principles, often implicitly, and use them when writing, animating, or acting out scenes. We must express them more explicitly in our agents, since the agents must exhibit these qualities on their own as action proceeds.

In previous papers we have presented lessons learned from our efforts to build believable non-linguistic agents. These include ideas on modeling emotion [3, 13], integrating emotion and action [4], real-time interactive personality animation [10], and a stand-alone framework for language generation [9]. As the transcript above suggests, building language-using believable agents raises new challenges.

In the rest of the paper we describe our approach to these challenges. We describe how we have extended Hap, a behavior-based architecture originally designed for producing action, to support natural language text generation; we discuss how we generate text and action on top of this enhanced architecture; and we analyze our progress toward responding to the challenges.

We see two reasons our work might interest other researchers, besides its direct contribution to creating language-using believable agents. First, it is a contribution to active language (also called "embodied language and action") research, and might suggest how other action architectures could be extended to support language generation. Second, the extensions to Hap to support language generation might be useful in expressing especially complex action generation behaviors.

2 Extending Hap to Support Language Generation

Existing Hap. Hap is a behavior-based architecture that we use as a foundation for building believable agents. Originally written in Lisp and designed to produce one action at a time, it was rewritten and extended when we built the Woggles to control multiple parallel motor and sensing channels in real-time [10].

In most of our agents, we build components on top of Hap to support emotion, social behavior, memory, and other functions. By building in Hap, these functions tend to inherit reactivity, parallelism, and other qualities that we want to permeate our agents. Our generic name for this composite framework is Tok. Thus, our goal can be seen as extending Tok (and in fact Hap) to support language generation.

We will explain the extensions we have developed using a minimum of Hap-specific terminology (but see [10] for further details). We hope thereby to help suggest how other behavior-based action architectures might be extended to support the production of language.

The relevant Hap concepts are as follows. Hap programs are written as collections of behaviors. Each behavior is something like a procedure, in that it is has a name (called the "goal"), a parameter list, a body, and a precondition. The body, in the simplest case, is a sequential or parallel invocation of other goals and primitive actions. When a goal is invoked, all behaviors named by that goal are considered for instantiation. Of the behaviors whose precondition is true, one is chosen and instantiated (with actual parameters for formals) in the hope of achieving the invoking goal. Thus, execution of a Hap program results in a tree of active goals, behaviors, subgoals, sub-behaviors, etc., some of which run as parallel threads.

Hap also supports multiple top level goals (ie, a forest of active behaviors), backtracking, various annotations to enhance reactivity in the face of surprises in the world, and other mechanisms. We have found these mechanisms useful for action, perception, emotion, and elsewhere in Tok, as well as for language generation. However, to keep this paper accessible, we defer to the first author's dissertation further discussion of these mechanisms and their relationship to language.

After the Woggles were developed, we wanted to extend Tok to begin to support

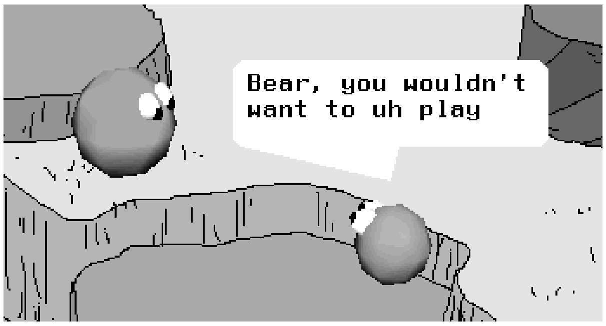

Figure 2: Woggles that "speak" text bubbles.

language. As a goal, we decided to create animated creatures that could generate (and understand) text bubbles, as suggested in Figure 2.

Glinda is a language generation architecture that had been created in our group [9]. It is a stand-alone system, but was developed with an awareness of Hap. We felt we saw similarities in the way Glinda made language generation decisions and the way Hap made action generation decisions. It seemed that by extending Hap a bit, we could make it easy for agent builders to express language generation behavior directly as a variety of action generation behavior. Thus, we chose to Glinda as our conceptual framework, and set about extending Hap to support it.

Our first step was to better support the representations that Glinda manipulated, which were far more structured than those Hap had typically handled. Then we considered ways to encode Glinda's grammar and other processing rules as Hap behaviors. This led us to introduce new ways of passing complex structures as parameters, accessing and matching structures, and passing information between widely separated behaviors. We will consider these, beginning with representation.

Glinda's main representational elements are *group*s and *feature*s. Features are name/value pairs, for example, `(root love)` and `(number singular)`. A group is a set containing features, subgroups, or both[2]. Groups must contain a `type` and a `role` feature. The type provides information about the expected elements of the group and the level of the group in the linguistic hierarchy, for example morpheme, word, phrase, or clause. The role specifies the group's function within its parent group. All subgroups within a group must have distinct roles.

As in Hap, Glinda execution proceeds via a gradual elaboration of goals. A Glinda goal is a group denoting a concept to be communicated together with a set of features that modify the generation process. The feature set is treated as part of the group, but it is convenient when invoking generation to be able to vary these additional features as an independent parameter.

A sample goal is shown in Figure 3. This goal would generate the sentence

[2]In our adaptation of Glinda, a group is very much like a frame or association list. However, we will retain the Glinda terminology to emphasize the mapping from language generation concepts.

```
group: ((type sentence) (role sentence)
        ((type relation) (role matrix)
         ((type parameter) (role predicate)
          ((type word)
           ((type morph) (cat verb) (transitive t) (root eat))))
         ((type parameter) (role agent)
          ((type word)
           ((type morph) (cat noun) (root john_1)
            (person 3) (number singular) (gender male))))))

features: ((mood declarative))
```

Figure 3: Example Glinda Goal

"John eats." If the features `(voice passive)` and `(time past)` were added to the features list (or the group) then it would yield "Something was eaten by John."

Representation. How shall we represent groups and Glinda goals in Hap? As mentioned earlier, Hap goals consist of a name and actual parameters. Since the values passed via a goal into a behavior are unrestricted, and since Hap's behavior language is Turing complete, we could simply represent Glinda goals as structured values passed into Hap behaviors. The components of the Glinda goals could be accessed by code written in Hap's existing behavior language, and this code could emulate Glinda's processing behavior. To a first approximation, this is the approach we adopted. However, accessing information inside Glinda goals is particularly unpleasant in this simple scheme. Thus, we chose to attempt a more fundamental merging of Glinda's and Hap's notions of goal.

Information in Glinda goals is accessed in characteristic ways. Most generation decisions are made by examining top-level features and subgroups. Features are accessed by name and subgroups are accessed by role. For example, in the goal in Figure 3, the fact that the `mood` is declarative and no `voice` feature is present might cause the `agent` subgroup to be the one first converted to text. When features are accessed, Glinda looks for them in the features list and in the goal. More deeply nested features and groups are seldom accessed. The exception is the *projector* of a group. The projector is a value of a feature deep within the group that forms the core concept of the group. For example, in a group of type sentence the projector is the verb, a feature which may be deeply nested. (Projectors are accessed only when determining which generation rules to apply, and we support them by providing an extension to the match language for preconditions.)

Because of these characteristic access patterns, we chose to support Glinda goals by adding first-class environments to Hap. An environment is a set of name/value bindings. As first-class entities, environments can be passed and stored as values. When an environment is passed as an actual parameter, it can be bound to a behavior's corresponding formal and accessed in the usual ways. However, an environment also can be *imported* (by using the particular formals named `group` and `features`). If imported, all of the bindings in the passed environment are added to the behavior's lexical environment and can be accessed like normal bindings.

In Hap, the values of parameters and variables are accessed by the form $$<variable>. We extend this to allow a variable's *binding*, to be accessed by the form $$&<variable>. A binding can be constructed using the form `(name value)`, where `value` is a value denoting expression. An environment value can be created by enclosing a sequence of bindings in parentheses.

A feature is now represented as a binding, and roughly speaking, both the group and set of features that comprise a Glinda goal are represented as environments. When these two environments are passed as values to a `generate` behavior, they are both *imported* so that all of their features and subgroups can be accessed by na¡me or role as normal Hap variables. More precisely, in order to be accessed by role, a group is represented as a binding whose value is an environment. The name of the binding is the role of the group and the environment contains all of the other information. When a group is imported, the bindings inside the environment are imported. In addition, a new binding is created and imported with name `role` and value the role (that is, the binding name) of the group.

Groups and features can now merge seamlessly with Hap goals and parameter passing. For instance, the following behavior shows how a natural language `generate` goal can be created:

```
(sequential_behavior request_play (who object)
  (subgoal generate
          (sentence ((type ^sentence) (hearer $$who)
                     (relation ((agent $$who) (predicate ^desire)
                                (object ((agent $$who)
                                         (predicate ^play)
                                         $$&object))))))
          ((focus ^play))))
```

This behavior tries to accomplish a `request_play` goal by generating natural language. The group and features are constructed using the binding and environment creating operators and include embedded variable accesses. The value of the `who` formal is included in the group in three places; the binding of the `object` formal appears once.

Processing. With this approach to representing Glinda groups, features, and goals, we can turn to processing. Glinda generation is performed by four types of rules: organization, combination, feature passing, and mapping. Organization rules take a group and set of features, select subgroups and features to be generated, and establish an order for them. For example, at the sentence level, organization rules choose between active voice order and passive voice order. At the word level, organization rules cause prefixes, roots, and suffixes to be generated. At the lexical level, they choose strings to realize items. In Hap this knowledge can be expressed as various sequential behaviors for `generate` goals. The Glinda test to determine if an organization rule applies in this situation can be written in the precondition for the behavior. An example behavior for passive voice is:

```
(sequential_behavior generate (group features)
       (precondition "$$type == ^relation && $$voice == ^passive")
  (with (bind_return_values_to subject_number subject_person)
    (subgoal generate $$&object))
  (subgoal generate $$&predicate
             ($$&inflection $$&subject_number $$&subject_person))
  (subgoal generate $$&agent))
```

This behavior can be used to instantiate a `generate` goal with type relation which contains a voice feature with value passive. It issues several `generate` sub-goals in order for the subgroups and features needed to express the group. Both the precondition and accesses to subgroups and features are concise because of the first-class environment representation for groups and features. These goals are each realized by other behaviors until eventually a series of strings is emitted.

Combination rules in Glinda smooth this sequence of strings. They introduce spaces between words and longer spaces after sentences, introduce no spaces between roots of words and their prefixes or suffixes, and perform character deletions, additions and substitution (such as "i" for "y" in "happiness"). A buffer is used to hold the most recent string generated and all applicable combination rules fire whenever a new string is generated. The (possibly modified) buffered string is then printed, and the new (possibly modified) string is placed in the buffer.

We support combination rules in Hap with a special `generate_string` behavior that is invoked only when a string is being generated. This behavior keeps the buffered string and new string in dynamically scoped local variables, creates a `combine` goal with annotations that force all applicable behaviors to run (these are Hap's `(persistent when_succeeds)` and `ignore_failure` annotations), and then prints the possibly changed old string and buffers the new string . Glinda's combination rules can then be written as normal Hap behaviors that match the buffered string and new string variables using preconditions and modify them through side-effects as desired.

Feature passing rules are Glinda's mechanism for communication between generation instances. They are used, for example, to express subject verb agreement and the coordination of tenses of root and auxiliary verbs. Upward feature passing rules filter at runtime the features that are returned after generation of a group. Downward feature passing rules determine which of these features is passed to the next subgroup to be generated. Each rule can pass any number of features, and all rules are evaluated to determine the set of features passed.

In Hap, features or values can be passed down to a subgoal as arguments to that goal. We added value returns to Hap behaviors to allow features or values to be

```
(parallel_behavior generate (group features)
      (precondition "$$type == ^location &&
                     query_looking_at($$hearer,$$me) &&
                     visible($$location)")
  (subgoal generate (pronoun ^location)
                    ((distance "distance($$location,$$me)")))
  (subgoal glance $$location))
```

Figure 4: Example of a language generating behavior.

passed up from subgoals to parent goals. This is a simpler mechanism than Glinda's, but one we hope will be adequate for our generation needs and easier to understand. In addition to allowing values to pass up and down the subgoal hierarchy, we provide another mechanism for communication between distant behaviors. Using dynamically-scoped variables and side-effects, one can create variables in a common parent and allow both goals to write and read from these variables. This eliminates the need for intermediate goals to explicitly pass along values. It also allows behaviors running in parallel to communicate through shared variables.

The final type of Glinda processing rule is the mapping rule. Mapping rules run just before an organization rule is chosen, and can add, delete or modify features that are present in the current group or set of features. They are used to perform simple kinds of inference. For example, a mapping rule to infer voice from focus is: if no voice feature is present and a focus feature has the same value as the actor then add `(voice ^active)`. Mapping rules are not supported in any special way in Hap. The inferences they perform can instead be encoded as normal Hap goals and behaviors, and called where appropriate.

3 Example of a Language Generating Behavior

Having described our extensions to Hap, let us consider Figure 4, which shows in simplified form how we might write a language producing behavior in extended Hap. This behavior encodes one of several possible ways to reference a location: by simultaneously gesturing and using a pronoun reference. Other methods to generate references to locations (when this behavior doesn't apply or is undesirable) are encoded in separate behaviors. The precondition states the conditions under which it is applicable: the group being generated must be of type location, the agent being spoken to must be looking at the speaker, and the location must be visible. The last two of these conditions are sensor queries that actively sense the world at the time the precondition is evaluated.[3] The behavior is parallel so that language production and gesturing occur simultaneously. The first subgoal generates the location pronoun subject to an auxiliary feature that encodes the distance to the location. This feature

[3]Primitive sensors can be embedded directly in match expressions, as in this precondition. More complex sensing is performed by sensing behaviors written in Hap.

is used to decide between realizations such as “there”, “over there”, or “over there about 30 feet”. The second subgoal causes the agent to look toward the location and then return its gaze to what it was tracking or looking at when the goal was issued.

4 Example of Processing

The code in Figure 4 suggests how we might write language and action producing behaviors. Let us now consider an execution trace of the processing that might result from full creatures built in this style. The creatures we discuss are Woggles [10] enhanced to use simple natural language. We hope to illustrate how our approach responds to the challenges for believability posed in the introduction.

As the trace begins, Shrimp has approached Bear who is looking away. Shrimp is sad, but wants to play a game with Bear at a nearby mountain. He decides to invite Bear to play, by invoking a `generate` goal for the group:

```
(sentence ((type ^sentence) (hearer $$who)
           (relation ((agent $$who) (predicate ^desire)
                      (object ((type relation) (agent $$who)
                               (predicate ^play)
                               (location $$where)
                               (object ^follow_the_leader))))))

((voice ^interrogative-yn))
```

As we will explain, this goal generates “Bear, you wouldn’t want [pause] uh [pause] to play over there, would you?”, while causing parallel action, sensing, etc.

The `generate` goal invokes a sequential behavior (the only sentence generating behavior in our current grammar) to perform the following subgoals:

```
(generate (punctuation ^beginning_of_sentence))
(generate $$&hearer)
(generate $$&relation)
(generate (punctuation ^end_of_sentence))
```

The first subgoal, when expanded, places a special symbol in the output buffer to mark the beginning of the sentence. This symbol cannot occur elsewhere in generation, and aids the capitalization and spacing combination rules.

The generation of the hearer feature has a number of behaviors from which to choose. If the generation of the example sentence is part of a larger on-going conversation between Bear and Shrimp, then a behavior would fire that would result in the empty string being generated for the hearer feature. Since that is not the case, a behavior is chosen to decide how best to get Bear’s attention. This is accomplished by sensing the world, for instance by making the first subgoal of this behavior be a sensing behavior to determine where Bear is and where he is looking. Since he is nearby but not looking at Shrimp, the behavior chooses to generate from the group `(name (object $$hearer))` followed by `(generate (punctuation`

`^end_of_pref))`, and issue the action to look at Bear in parallel. The first results in "Bear" being generated. When this happens the combine goal is posted with the buffer contents (beginning of sentence symbol) and the newly generated string "Bear". The combine goal persists until no behaviors apply for it. In this case there is only one behavior that applies. It removes the beginning of sentence symbol and capitalizes the first character of the newly generated string, a no-op in this case because "Bear" is already capitalized. The `(punctuation ^end_of_pref)` group results in a comma being generated. No combination behaviors fire for these two, so at this point "Bear" is printed, "," is in the buffer, and Shrimp is looking at Bear.

If Bear had been across the room, this generation behavior would have resulted in Shrimp looking at Bear and generating "Hey Bear!". Alternatively if Bear had noticed Shrimp's approach and was watching Shrimp attentively, a behavior would have been chosen to generate the empty string. Thus, sensing is being used to affect generation on a fine time scale.

The next goal to be executed is `(generate $$&relation)`. There are several ways to order the subgroups of a relation to be generated as a sentence, for example active voice order or passive voice order. In this case the voice is specified as `interrogative-yn`, but we could still choose a number of orderings (including active or passive word order followed by a question mark). Because Shrimp is currently feeling very sad (see emotion discussion below), a negative phrasing of the question is chosen. This results in the following sequence of subgoals:

```
(generate $$&actor)
(bind_return_values_to (modal)
  (generate $$&predicate ((negated t))))
(generate $$&object)
(generate_string ",")
(generate $$&modal)
(generate $$&actor).
```

Because the `actor` is also the `hearer`, the first subgoal is generated as "you". The hearer's identity is stored in a dynamically scoped variable in a parent behavior, and this is accessed and tested in the behavior for generating the actor.

The second subgoal is an example of explicit feature passing by returning which modal is used in the generation of the predicate. This information is used to generate the modal later in the sentence. This allows the same behavior to be used for "You don't ..., do you?" as for "You wouldn't ..., would you?". (The sentence "Wolf isn't ..., is he?" uses the same rule. The particulars of generating the `actor` subgroup changes.) In this case, the subgoal produces "wouldn't want" as output.

At this point, Shrimp notices that Wolf is coming toward him at high speed. He notices it via the firing of a higher level sensing goal. This knowledge gives Shrimp a good bit to think about, and the resulting processing elsewhere slows the Hap thread that is running this generation task. He becomes afraid. He actively looks to decide if he should run or get out of the way. Observers can notice that something is going on because Shrimp stops generating words. In addition, part of the behavior to communicate is a parallel goal that watches for pauses in output and inserts stuttering "uh"s at intervals during these pauses. This goal is only active when a communication goal is active. As Shrimp's pause becomes longer, this goal is triggered, and Shrimp says "uh". Shrimp continues to watch Wolf, and decides to move slightly to let him pass more easily. As Wolf goes by, Shrimp continues to generate, producing "to play".

Shrimp now generates the relation's `location` subgroup. There are several potential behaviors. Since Bear is looking at him and a `formal` feature is not present, a behavior is chosen to gesture and concurrently generate a pronoun referring to the location. So, Shrimp says "over there" as he glances toward the mountain.

Finally, the trace ends as the last three subgoals generate ", would you?".

To summarize the behavior that is observed in this example, Shrimp has just come over to Bear who has not noticed him. Shrimp starts looking at Bear at the same time he says "Bear, ". He goes on to say "you wouldn't want" one word at a time, when he pauses and looks over at Wolf racing toward him. He looks around, says "uh", moves slightly to get out of Wolf's way and continues to say "to play a game". As he says the next two words "over there" he glances toward the mountain. Looking at Bear again, he concludes with ", would you?".

5 Discussion of Results

The above trace suggests how our system responds to the challenges raised in the introduction. Let us consider our attempt to meet them in more detail.

Incremental language generation is a property of Glinda that we have maintained in our approach. Pauses, restarts and other breakdowns due to the difficulty of the generation task itself are visible in Glinda and in our system. However, with the generation process expressed as Hap goals and behaviors in an agent with other goals and behaviors, pauses or other breakdowns due to other aspects of the agent can arise and be visible. These include pauses caused by the agent attending to goals activated by events in the external world (e.g. Wolf's approach in the example) as well as goals triggered by internal events. For example, generation could infer a piece of information, which when placed in memory awakens an otherwise independent goal. This goal might then perform more inference, generate emotions, generate action or language, etc. This might be similar to the types of pauses people make when realizing something while talking. The pause caused by this thinking would be visible in the timing of text output. Any actions or language produced by this digression would cause the mental digression to be even more visible.

Responsiveness to the environment and reactivity are properties of the system (Tok) that Hap provides. As just mentioned, with generation expressed in Hap, generation can be interrupted at any point while the agent deals with other goals. Hap behaviors are *situated* in the sense described by Agre and Chapman [1, 2] in that they are interpreted in the context of the current situation. The amount that particular behaviors take advantage of being situated is dependent in part on the amount of sensing they include. We are attempting to take advantage of behaviors being situated by constructing our generation behaviors (that is, our grammar) to include sensing. This sensing can be external as when Shrimp looks at Bear to decide how to refer to a location and whether to introduce his question with Bear's name, or the sensing can be internal as when Shrimp's emotional state is sensed to choose a negative phrasing of the question.

Just as we can invoke sensing within generation, action subgoals can be included in generation behaviors. Though we are just beginning to explore this topic, we have thus far found it natural to mix these three types of processing to accomplish the agent's goals. Some of that naturalness, we hope, is conveyed in the trace.

Hap maintains multiple active goals and pursues them concurrently. This is done by using the processing power available, when a behavior is paused, to pursue other goals. For example, once a Woggle jumps, it does not have to consider that goal until its body is almost ready to land. Likewise, after asking a question, an enclosing behavior need not be attended to until a response is heard or too much time elapses. During these pauses, Hap attends to other (perhaps unrelated) active goals. If these goals issue actions, the actions will occur in parallel with other executing actions. The actions (and goals they are in service to) must not use the same resources that other executing goals are using. The primitive action to print text in a voice bubble executes in time proportional to the length of the string.[4] Since generation goals are normal Hap goals, this same mechanism allows other unrelated goals to be pursued in parallel. The saying of "uh" while moving aside in the example is an instance of two unrelated goals issuing actions that are executed in parallel.

This same mechanism is used to simultaneously generate action and language to accomplish a communication goal. For example, gesturing toward the mountain and saying "over there" in the trace was accomplished by a parallel behavior with two subgoals: an action goal to perform the gesture and a generation goal to realize the text. One of these subgoals was chosen arbitrarily to be pursued. When the action was initiated and that goal was suspended waiting for the action to (nearly) complete, the other goal was pursued, issuing the parallel action.

Integrating emotion with language generation is accomplished in the same way emotion is integrated with action. The emotional state of the agent is available to all behaviors and can be used to include emotion-based variation. One example of this is illustrated in the example when Shrimp chooses a negative phrasing of his question over the more straightforward phrasing. How to introduce meaningful emotion-based variation in language in general is of course a difficult problem, and we are drawing on existing work, for example Hovy's thesis work [8], where possible in pursuing it. The fact that the generation process is embedded in an actual

[4]This is to model the fact that speaking and typing both take time to execute, even after what is to be typed or said is known.

agent with emotions gives us a rich base on which to explore this issue.

As with other goals, success and failure of `generate` goals can cause emotion. This happens automatically if a character builder marks goals that are emotionally important. Success or failure of these goals produces joy or distress. Anger or gratitude arise if the cause of success or failure can be inferred. This is often done by inference goals that are written and issued in parallel with the `generate` goal. For further description of the emotion model and its use in Tok see [4].

6 Related Work

Chapman's thesis work on Sonja [7] and Firby and Martin's work integrating RAP with DMAP [11] have similarities with our work. Both systems tightly integrate language understanding with action generation and make use of the situated nature of the tasks. They do little or no generation, however, and do not address some of our goals, such as creating believable agents, displaying the internal state of the agent through pauses, emotion and personality based variations, etc.

Rich, et al [14] seem to share our goal of building engaging, believable agents, and they have integrated a remarkably wide range of capabilities in their system, including language understanding, speech synthesis and recognition, low-level motor control, and simple cognition and emotion. Their broad integration takes a traditional approach in that there are distinct components for each of the capabilities and they communicate through relatively narrow channels. This is a natural consequence of building upon existing, independently developed components. However, our impression from using their system is that the resulting agents generally lack an appearance of awareness that we feel is crucial for our goals of believability.

Recent work in speech synthesis and gesture by Cassell et al. [6] and speech synthesis and facial animation by Nagao and Takeuchi [12] is relevant to our goals. This work offers insight into fine-grained timing and interaction of action and speech. However, like Rich, it takes a less integrated approach than the one presented in this paper, with some of the same limitations for believability.

Rubinoff and Lehman, in NL-Soar [15], share our goal of real-time, responsive language generation mixed with action. They take a modular approach to integration, like the systems above, but through learning NL-Soar gradually becomes tightly integrated. This lets them develop language somewhat independently of the rest of their agent, yet still get integration such as language using the agent's sensing and inference abilities to accomplish its goals. Because their task domains involve communication by radio, they have not pursued coordinated action and language to accomplish a single communication goal. Also, their agents as yet have no emotion model, so they have not explored this aspect of integration. Finally, the ultimate goal of NL-Soar is cognitive plausibility rather then believability. Nonetheless, of the efforts reported here, we see this work as most closely related to our own.

7 Conclusion

We have described an approach to creating believable agents that act and generate natural language text in a simulated world. This includes a description of extensions to Hap, our behavior-based action architecture, to better support natural language generation, and a description of how the resulting system is used to generate action and language. We believe these extensions may be useful to others developing similar believable agents, and to researchers interested in extending action architectures with language capabilities.

Prior to extension, Hap provided facilities we used extensively in our approach to language generation. These include hierarchical (and recursive) invocation of behaviors with parameterized goals, selection by pattern matching of behaviors to achieve a goal, determination of success and failure of goals by means other than testable conditions, and concurrent threads pursuing parallel goals (and subject to resource constraints). In our experience, these are important capabilities in behavior-based architectures that are intended to support complex action, such as language.

On this foundation, we extended Hap by adding first-class environments, allowing behaviors to return values, providing dynamic scoping with side-effects (in addition to the previously existing lexical scoping), and building a `generate_string` behavior to support Glinda's notion of combination rule. As argued in the Discussion of Results section above, we believe the resulting framework, and our use of it to produce natural language text, address in significant ways each of the challenges raised in the introduction.

While we feel that the challenges raised in this paper are important for believability, the full requirements for believability remain little understood. Gradually the task should become more clear, through further research and gatherings such as the AAAI 94 Spring Symposium on Believable Agents, Lifelike Computer Characters '94, the believable agents session at AAAI-94, and the upcoming AAAI 95 Fall Symposium on Embodied Language and Action. We believe the work reported here will help us move toward the ultimate goal of believable interactive characters.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, July 1987.

[2] Philip E. Agre and David Chapman. What are plans for? In *Robotics and Autonomous Systems*. Elsevier Science Publishers, 1990.

[3] Joseph Bates. The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125, July 1994.

[4] Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. Integrating reactivity, goals, and emotion in a broad agent. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, IN, July 1992.

[5] *Casablanca.* Warner Brothers, Inc. Avail. from Turner Entertainment., 1942.

[6] Justine Cassell et al. Animated conversation. In *Proc. SIGGRAPH '94*, 1994.

[7] David Chapman. *Vision, Instruction, and Action*. PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1990.

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

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

[10] A. Bryan Loyall and Joseph Bates. Real-time control of animated broad agents. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO, June 1993.

[11] Charles E. Martin and R. James Firby. Generating natural language expectations from a reactive execution system. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 1991.

[12] Katashi Nagao and Akikazu Takeuchi. Social interaction: Multimodal conversation with social agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

[13] W. Scott Reilly and Joseph Bates. Building emotional agents. Technical Report CMU-CS-92-143, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1992.

[14] Charles Rich et al. An animated on-line community with artificial agents. *IEEE MultiMedia*, 1(4):32–42, Winter 1994.

[15] Robert Rubinoff and Jill Fain Lehman. Real-time natural language generation in NL-Soar. In *Proceedings of 7th Internat. Generation Workshop*, June 1994.

[16] Frank Thomas and Ollie Johnston. *Disney Animation: The Illusion of Life*. Abbeville Press, New York, 1981.