Personality-Rich Believable Agents That Use Language

A. Bryan Loyall and **Joseph Bates**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
bryan.loyall@cs.cmu.edu, joseph.bates@cs.cmu.edu

Abstract

We are studying how to create believable agents that perform actions and use natural language in interactive, animated, real-time worlds. Believable agents are autonomous agents that have specific, rich personalities like characters in movies and animation. We have extended Hap, the behavior-based architecture used by the Oz group to construct non-linguistic believable agents, to support natural language text generation. These extensions allow us to tightly integrate text 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; varying generated text to express the specific personality; 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.

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 or art (Bates 1992; Hayes-Roth *et al.* 1995), 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 (casablanca 1942), 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.

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.

In these moments, Ugarti is a very believable and engaging

Agents '97 Marina del Rey CA USA
(c) 1997 ACM

[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 (Thomas & Johnston 1981).

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 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 the rest of the paper we describe our approach to these challenges. We describe how we have extended Hap (Loyall & Bates 1993), 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.

In addition to its direct contribution to creating language-using believable agents, we see two reasons our work might interest other researchers. First, it is a contribution to active language (also called "embodied language and action") research[2], 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 in behavior-based architectures.

Extending Hap to Support Language Generation

In this section we describe how we have extended Hap, a behavior-based architecture designed for nonlinguistic believable agents, to directly support natural language generation. Previously, Hap has been used to create a number of prototype believable agents. These include a simulated house cat named Lyotard (Bates, Loyall, & Reilly 1992a), and three real-time, self-animating agents called Woggles in a work titled "Edge of Intention" (Loyall & Bates 1993). Edge of Intention was first shown at the AAAI-92 AI-based Arts Exhibition and then subsequently at SIGGRAPH-93, at Ars Electronica '93, and in the Boston Computer Museum. Informal evidence of people interacting with these agents (hundreds in the case of the Woggles) suggest that they are somewhat successful as believable agents with distinct personalities.

Hap is a behavior-based architecture developed to extend the ideas of situated activity and reactivity (Agre & Chapman 1990; Brooks 1986) with explicit goals, which we have found useful for believable agents. Hap is similar to other reactive architectures (Firby 1989; Georgeff & Lansky 1987; Nilsson 1992; Simmons 1991; Maes 1989; Kaelbling & Rosenschein 1990), and we suspect the approach we present here to support generation could be adapted to many of these architectures.

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 (i.e., 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 Hap, as well as for language generation.

[2] Interest in this area has been growing in recent years as evidenced by the AAAI 1994 Spring Symposium on Active Language and the AAAI 1995 Fall Symposium on Embodied Language and Action.

In most Hap agents, components on top of Hap are built to support emotion, social behavior, memory, and other functions. The emotion model itself was developed by Reilly and Bates (Neil Reilly 1996). By building in Hap, these functions tend to inherit reactivity, parallelism, and other qualities that we want to permeate our believable agents. For a more detailed description of Hap and higher functionality built in Hap please see (Loyall & Bates 1993).

We will explain the extensions we have developed using a minimum of Hap-specific terminology. We hope thereby to help suggest how other behavior-based action architectures might be extended to support the production of language.

We chose to base our language approach on the Glinda generator developed by Kantrowitz and Bates (Kantrowitz 1990; Kantrowitz & Bates 1992) for three reasons. First, Glinda is an *integrated* generator (Kantrowitz & Bates 1992), with the same generation engine used for both text planning and realization, and we saw similarities between the Hap execution engine and Glinda's generation engine that suggested that a fundamental merging might be possible. Second, Glinda outputs text incrementally, which we believe important for real-time believable agents. And third, Kantrowitz's research is concerned with exploring pragmatic variation for natural referring expressions. In the future we may want to incorporate this work to allow our agents to generate more natural text.

There are two main challenges to expressing text generation in an action architecture such as Hap. The first challenge is solely a knowledge representation issue. Glinda's generation goals are significantly more structured than those normally handled by Hap. In order to pursue generation in this action architecture we had to extend it to handle the central properties of Glinda's structured goals. The second challenge is how to allow the encoding of the grammar. Glinda includes four different types of processing knowledge as rules that fire in stages. In an action architecture such as Hap we must express this knowledge in terms of goals and behaviors in some way that does not make encoding this knowledge onerous. In addition, we must do this encoding without compromising properties of Hap that are important for believability, for example Hap's reactivity, and real-time properties (Loyall & Bates 1993).

The basic approach we are taking to generation is hierarchical decomposition: given a structured concept to be generated, break it into subconcepts to be generated and issue each of those in turn as subgoals of a sequential behavior. This is the same basic approach Hap uses for action generation[3], and that many natural language generation systems including Glinda use.

We extended Hap to support the same data structures for concepts that Glinda uses, allowing the use of normal Hap goals and sequential behaviors for this hierarchical expansion. Multiple, competing, behaviors with appropriate preconditions encode the linguistic knowledge to decompose and order concepts appropriately.

These behaviors operate at multiple levels in the generation process. At the text planning level, they select which concepts to convey and in what order to present them. At the realization level they follow appropriate precedence relations, and at the word level they order morphemes.

```
    group: (receiver (type ^relation)
                     (time ^present)
                     (agent (type ^agent) (value ^bear))
                     (predicate (type ^action)
                                (value ^play)))
features: ((case ^objective) (hearer ^bear))
```

Figure 2: Example Concept to Generate

The data structures used by most generators including Glinda are far more structured than those Hap had typically handled. Hap goals are a flat list of a name and zero or more values. In contrast, a concept to be generated in Glinda is expressed as a structured *group* and set of *features*. Groups and features are expressively the same as frames or association lists. A feature is a name/value pair, and a group is a *role* followed by an unordered collection of groups or features. An example group and set of features is shown in Figure 2. This concept represents a play relation with the agent Bear as the actor. Its role is the receiver of the action in an enclosing relation. When generated, it produces the dependent clause "that you play" or "to play" depending on the verb of the enclosing relation. Pronominalization or elision of the agent occurs in this case because the agent of the relation is the same as the one being spoken to (as indicated by the `hearer` feature).

To effectively support groups and features for this processing, we had to extend Hap to support four key properties of the representation:

1. Groups and features need to be easy to create, and must provide easy access to component subgroups and features. In this NLG model, nearly all accesses to the group are to its immediate subparts, but see item 3 below.

2. The group and features set are independently specified when invoking a generation goal, but features in the feature set are treated as if they are included in the group. Any references to features must look in both the group and features set. This property allows the subgoaling process of generation to choose to generate a subcomponent of the current group and pass separately any modifying information through features. With the features in both the group and features set treated identically, modifying features can be embedded in the group or subgroups if they are important to the meaning and known ahead of time, or they can be included easily in the feature set that is created in the process of generation.

3. The only nested portion of a group that is accessed is its *projector*. Conceptually, the projector is the core concept of the group, and along with the type and role of the group, it is central in choosing how the group is generated. It is found by a recursive traversal of the group, choosing the most central subgroup or feature at each level. When a feature is chosen, the value is returned as the projector of the group. The role of the most central component for each group type is specified in advance. For example, the most central component of a group of type relation is the subgroup with role "predicate", and the most central component of a group of type word is the "root" component.

[3] With other embellishments such as parallelism, situated variation, and reactive annotations.

4. The final property of the representation is a global abstraction hierarchy for matching symbols within a group. It is often useful to write general rules that apply to all groups with a projector that is a verb, for example, with more specialized rules applying to groups with a projector that is an auxiliary verb or a specific verb. With an abstraction hierarchy applying to all of these matches, these general and more specific rules can be easily written.

In our implementation, we supported these properties by adding first-class environments to our action architecture to support easy creation of groups, features and sets of features as well as the uniform access to elements of a group and set of features (properties (1) and (2)). We support property (3) and (4) by extending the matching language that all Hap agents use for preconditions and reactive annotations. We have found that these extensions allow us to manipulate these data structures effectively for performing generation.

With these data structures supported, we can turn our attention to the second challenge of allowing the encoding of the four types of generation process knowledge without compromising Hap's properties. As described above, the first type of knowledge, the hierarchical expansion that is the core of the generation process, can be naturally expressed as normal Hap goals and sequential behaviors.

The eventual result of this expansion is a sequence of strings over time. Local modifications are often necessary between pairs of these strings, for example to 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").

Combination rules perform these functions in Glinda. 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 this knowledge 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. Rules for these local modifications between strings 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.

A number of effects in generation require communication between generation subgoals, for example subject-verb agreement and the propagation of verb tenses. In Glinda this knowledge is encoded in a third category of rules called *flow rules*. All rules of this type are evaluated before a generation goal is pursued. In our approach to text generation in Hap we chose not to manage information flow using rules for two reasons. First, Hap already has mechanisms for information flow using normal parameter passing and value returns. And second, using rules for information flow introduces potential responsiveness problems for the agent. If used excessively the time required to evaluate these rules might compromise responsiveness of Hap agents by introducing arbitrary delays between goal expansions.

```
(parallel_behavior  generate (group features)
       (precondition  $$type is a ˆlocation and
                      $$hearer is looking at me and
                      $$location is visible )
       (context_condition  $$hearer is looking at me until
                           subgoal 2 is done )
   (subgoal generate  pronominal reference to $$location )
   (subgoal glance $$location))
```

Figure 3: Example of a language generating behavior.

For these reasons, we facilitate such information flow through two mechanisms. The first is normal parameter passing and value returns from subgoals. The second mechanism uses dynamically-scoped variables and side-effects. We should point out that dynamic variables are not needed to encode the information flow necessary for generation; this information flow can be expressed using Hap's normal lexically-scoped parameter passing and value returns. Nevertheless, we have found that dynamic scoping can provide certain advantages when carefully used. One can create dynamic variables in a common parent and allow both communicating goals to write and read from these variables. This eliminates the need for intermediate goals to explicitly pass along values. Dynamic scoping can also be used to allow behaviors running in parallel to communicate through locally shared variables.

Both of these mechanisms have the advantage over rules for information flow in that minimal processing is necessary at runtime to execute them. Thus far they have been expressive enough to encode the information flow necessary for generation.

The final type of processing needed for generation is inference. Glinda allows these inferences to be encoded as a fourth type of rule. All rules of this type are evaluated after flow rules and before a generation goal is pursued. Inference in Hap is done using normal Hap goals and behaviors. Appropriate inference goals can be included in generation behaviors wherever they are needed.

Example of a Language Generating Behavior

With the extensions described, one can express a traditional grammar in Hap directly. Generation then automatically inherits properties from Hap we believe important for believable agents. In addition, this combined architecture allows the agent builder to express additional knowledge in the grammar such as: how sensing affects make linguistic choices, how emotions influence generation, or how the production of language should react to changes in the world. Later sections illustrate and describe these implications of the integration in more detail. Figure 3 shows in simplified form how one such language generation behavior is expressed. This behavior encodes one possible way 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

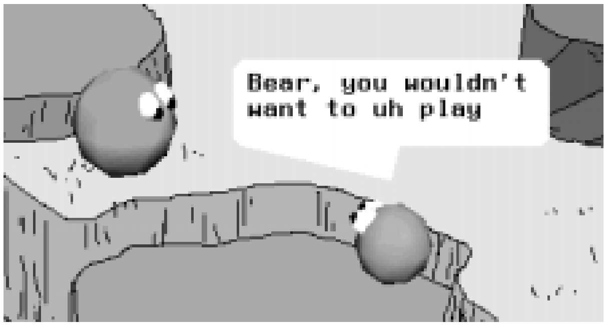

Figure 4: Woggles that "speak" text bubbles.

generated must be of type location[4], 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.[5] Action and language are mixed in this behavior simply by including both goals as subgoals of the behavior. These goals are pursued concurrently because the behavior type is parallel rather than sequential. To react to changes that might occur while this behavior is executing, appropriate annotations can be added. In this case it is important that the hearer doesn't look away before the gesture is performed. This is captured by adding a context condition with this information to the behavior. This condition causes sensing to be performed while this behavior is active. If the condition becomes false, this behavior is aborted, and another (perhaps referring to the location by name) will be chosen.

Example of Processing

We have tested this approach to language by implementing the extensions to Hap and building behaviors for a particular agent that incorporates language generation as part of its normal activity. The agent we have extended is Shrimp, one of the characters from the Woggles (Loyall & Bates 1993). A snapshot of this interaction can be seen in Figure 4. As suggested by this figure, the generated text appears, as it is generated in real-time, in a text bubble that is positioned above the woggle that is generating it. The other woggle can be either another autonomous agent from the world or the user woggle, which is controlled by a human using the mouse and keyboard. In either case, it can interact in real-time using physical motion, including motion of the eyes, as well as language (interpreted by keyword matching limited now to various versions of "yes" and "no").

To illustrate how our approach responds to the challenges for believability posed in the introduction, let us now consider a detailed example of an interaction with this agent.

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

```
(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 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)`. The mood is specified as

[4] Because we support groups using first-class environments, feature values are accessed just as normal Hap values; this is noted syntactically by prefixing the name with "$$". The complete feature (name and value) is accessed by prefixing the name with "$$&". Subgroups are accessed using the same notation.

[5] Primitive sensors can be embedded directly in match expressions, as in this precondition. More complex sensing is performed by sensing behaviors written in Hap which can be included as subgoals of the behavior.

interrogative-yn, but there are still several ways to order the subgroups to be generated as a sentence. 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 the aggressive creature, 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?”.

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.

Hap’s pursuit of multiple goals concurrently preserves the overall responsiveness of the agent while it generates language. As just mentioned, with generation expressed in Hap, generation can be interrupted at any point while the agent pursues other goals. These interruptions are managed by the relative priorities of the goals of the agent. Of course, it is possible to write computations, in language or elsewhere, that undermines this responsiveness, for example writing a high-priority infinite loop. The speed of the generation process itself is a product of how the grammar is written, for example how much inference is included. This speed should be appropriate to the personality being built; a thoughtful character might have more pauses in his speech than an impulsive character.

In our architecture, sensing can be used for any linguistic decisions desired by the character builder. This allows these grammars to be *situated* in the sense described by Agre and Chapman (Agre & Chapman 1990) in that they 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 use sensing within generation, action subgoals can be included in generation behaviors. Others have looked at this in the context of effective communication (Appelt 1985). In believable agents, the focus needs to be not just effective communication but also the expression of personality. Some characters include action with their language more than others. And for some whether to include guestures or not is a matter of their mood: when happy they guesture, when angry they move very little. Hap was created for such personality-based variation, and it is one of our goals when exploring the inclusion of action with language.

Reactivity is possible during generation just as it is in other Hap behaviors. Annotations can be associated with behaviors and goals to recognize when the situation changes enough that the behavior should change. For example, in the example Shrimp chose the negative phrasing because he was sad. If he becomes slightly happy while talking, this phrasing is no longer appropriate. This can be recognized by a context condition for this behavior that recognizes when he becomes happy. With this condition, if Shrimp becomes happy while the behavior is active, it would be aborted and another (more appropriate one) could be chosen for the goal. This would result in language such as: "Bear, you wouldn't want ... hey, let's play!" Other reactive annotations can recognize the spontaneous achievement of goals (such as Bear interrupting Shrimp to say "sure"). A success test that recognizes this allows Shrimp to stop talking and the two of them to start playing immediately. Either type of annotation can be included at any level of the grammar.

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.[6] 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 (Bates, Loyall, & Reilly 1992b). 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 in emotional and socially-based language variation, for example (Hovy 1988; Walker, Cahn, & Whittaker 1996), where possible, in pursuing it. The fact that the generation process is embedded in an actual 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 these agents see (Neil Reilly 1996; Bates, Loyall, & Reilly 1992b).

Related Work

Chapman's thesis work on Sonja (Chapman 1990) and Firby and Martin's work integrating RAP with DMAP (Martin & Firby 1991) 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 and colleagues (Rich *et al.* 1994) 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 and colleagues (Cassell *et al.* 1994) and speech synthesis and facial animation by Nagao and Takeuchi (Nagao & Takeuchi 1994) are relevant to our goals. This work offers insight into fine-grained timing and interaction of action and speech. However, like Rich, they tend to take a less integrated approach than the one presented in this paper, with some of the same limitations for believability.

Similarly, the continuing work of Webber and Badler and colleagues (Badler *et al.* 1995) on human simulation and language use are relevant to our long-term goals of creating believable agents that use language and act in real-time worlds. And more recently they have been interested in personality in their agents, especially with work in social behaviors like hide and seek. However, like Chapman, Firby and Martin, they do little or no generation, and like Rich, Cassell and Nagao they take a less integrated approach than we think necessary for believability.

Rubinoff and Lehman, in NL-Soar (Rubinoff & Lehman 1994), 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 learn-

[6] 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.

ing 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.

Conclusion

We have described an approach to creating believable agents that act and generate natural language text in a real-time 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.

This approach addresses many of the challenges we see for believable agents that both act and generate natural language in real-time worlds. We have described these challenges and how our approach addresses them including: using language and action together to accomplish communication goals; using perception to help make linguistic choices; varying generated text according to emotional state; varying generated text to express the specific personality; producing emotions as the result of the success, failure and other processing of language generation; and issuing the text in real-time with pauses, restarts and other breakdowns visible.

References

Agre, P. E., and Chapman, D. 1990. What are plans for? In *Robotics and Autonomous Systems*. Elsevier Science Publishers.

Appelt, D. E. 1985. *Planning English Sentences*. Cambridge University Press.

Badler, N.; Metaxas, D.; Webber, B.; and Steedman, M. 1995. The center for human modeling and simulation. *PRESENCE: Teleoperators and Virtual Environments* 4(1):81–96.

Bates, J.; Loyall, A. B.; and Reilly, W. S. 1992a. An architecture for action, emotion, and social behavior. In *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*.

Bates, J.; Loyall, A. B.; and Reilly, W. S. 1992b. Integrating reactivity, goals, and emotion in a broad agent. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*.

Bates, J. 1992. Virtual reality, art, and entertainment. *PRESENCE: Teleoperators and Virtual Environments* 1(1):133–138.

Brooks, R. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2:14–23.

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

Cassell, J., et al. 1994. Animated conversation. In *Proc. SIGGRAPH '94*.

Chapman, D. 1990. *Vision, Instruction, and Action*. Ph.D. Dissertation, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Firby, J. R. 1989. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. Dissertation, Department of Computer Science, Yale University.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*.

Hayes-Roth, B., et al. 1995. Directed improvisation by computer characters. Technical Report KSL-95-04, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Hovy, E. 1988. *Generating Natural Language under Pragmatic Constraints*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Kaelbling, L. P., and Rosenschein, S. J. 1990. Action and planning in embedded agents. *Robotics and Autonomous Systems* 6(1-2):35–48.

Kantrowitz, M., and Bates, J. 1992. Integrated natural language generation systems. In Dale, R.; Hovy, E.; Rosner, D.; and Stock, O., eds., *Aspects of Automated Natural Language Generation*, volume 587 of *Lecture Notes in Artificial Intelligence*, 13–28. Springer-Verlag. (This is the Proceedings of the Sixth International Workshop on Natural Language Generation, Trento, Italy, April 1992.).

Kantrowitz, M. 1990. 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.

Loyall, A. B., and Bates, J. 1993. Real-time control of animated broad agents. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*.

Maes, P. 1989. The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*.

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

Nagao, K., and Takeuchi, A. 1994. Social interaction: Multimodal conversation with social agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*.

Neil Reilly, W. S. 1996. *Believable Social and Emotional Agents*. Ph.D. Dissertation, Computer Science Department, Carnegie Mellon University.

Nilsson, N. J. 1992. Toward agent programs with circuit semantics. Technical Report STAN-CS-92-1412, Department of Computer Science, Stanford University, Stanford, CA.

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

Rubinoff, R., and Lehman, J. F. 1994. Real-time natural language generation in NL-Soar. In *Proceedings of 7th Internat. Generation Workshop*.

Simmons, R. 1991. Concurrent planning and execution for a walking robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Thomas, F., and Johnston, O. 1981. *Disney Animation: The Illusion of Life*. New York: Abbeville Press.

Walker, M. A.; Cahn, J. E.; and Whittaker, S. J. 1996. Linguistic style improvisation for lifelike computer characters. In *Entertainment and AI / A-Life, Papers from 1996 AAAI Workshop*. Available as AAAI Technical Report WS-96-03.