A Behavior Language: Joint Action and Behavioral Idioms

Michael Mateas[1] and Andrew Stern[2]

[1] College of Computing and School of Literature, Communication and Culture
Georgia Institute of Technology
`michaelm@cc.gatech.edu`
`www.cs.cmu.edu/ michaelm/`

[2] InteractiveStory.net
`andrew@interactivestory.net`
`www.interactivestory.net`

Summary. This chapter presents ABL (A Behavior Language, pronounced "able") , a language specifically designed to support the creation of life-like computer characters (believable agents). Concurrent with our development of ABL, we are using the language to implement the believable agent layer of our interactive drama project, *Façade*. With code examples and case-studies we describe the primary features of ABL, including sequential and parallel behaviors, joint goals and behaviors for multi-agent coordination, and reflective programming (meta-behaviors). Specific idioms are detailed for using ABL to author story-based believable agents that can maintain reactive, moment-by-moment believability while simultaneously performing in tightly coordinated, long term dramatic sequences.

1 Introduction

ABL is based on the Oz Project (Bates [1]) believable agent language Hap, developed by A.B. Loyall (Loyall and Bates [17], Bates, Loyall, and Reilly [2]). Hap was designed to support the detailed expression of artistically-chosen personality, automatic control of real-time interactive animation, and architectural support for many of the requirements of believable agents (Loyall [15]).

ABL extends Hap in several ways, most significantly by adding joint goals and behaviors, which support the multi-agent coordination required for the performance of dramatic action. ABL also changes Hap's syntax, making it more Java-like, and generalizes the mechanisms by which an ABL agent connects to a sensory-motor system, making it possible for others to use ABL in their own believable agent projects.

This chapter will discuss ABL by means of examples from *Façade*, as well as an early Oz believable agent project, the *Woggles* (Loyall and Bates [16]).

2 Why ABL?

Believable agents are applicable as non-player characters in interactive stories and games, as tour guides through virtual spaces, teachers and tutors in educational software, virtual salespeople for marketing products, and so on. To achieve a non-trivial degree of life-likeness in such agents, they must possess the ability perform several intelligent activities in parallel – for example, to gaze, speak, walk, use objects, gesture with their hands and convey facial expressions, all at the same time. Additionally, while performing these parallel behaviors, believable agents need to be reactive in immediate, varied and fine-grained ways, so as to respond convincingly and satisfyingly to the user's moment-by-moment interaction. In ABL, an activity (e.g., walking to the user, or speaking a line of dialog) is represented as a goal, and each goal is supplied with one or more behaviors to accomplish its task. An active goal chooses one of its behaviors to try. A behavior is a series of steps, that can occur sequentially or in parallel. Typically, once a behavior completes all of its steps, it succeeds and goes away. However if any of its steps fail, then the behavior itself fails and the goal attempts to find a different behavior to accomplish its task, failing if no such alternative behavior can be found. Furthermore, a behavior may have subgoaled its own set of goals and behaviors. To keep track of all the active goals and behaviors and subgoal relationships, ABL maintains an *active behavior tree* (ABT).

In contrast to standard imperative languages one might use to control agents (e.g. C++, Java), in ABL an author can, in relatively few lines of code, specify collections of goals and behaviors that can cleanly inter-mix character actions, modulate their execution based on the continuously sensed state of the world, and perform local, context-specific reactions to a player's actions.

This paradigm of combining sequential and parallel behaviors, and propagating success and failure through the ABT, are the foundation of the power of ABL as a language for authoring believable agents. Parallel behaviors make it easy to author characters that pursue multiple goals and thus mix the performance of multiple behaviors. This powerful capability doesn't come for free – it effectively makes ABL a parallel programming language, thus introducing to the author the well-known complexities of parallel programming. ABL is designed to make simple character behavior easy to author with just a few lines of code, while still providing the power to let experienced authors write complex, expressive behavior. ABL's support for *joint* goals and behaviors helps the author to harness the expressive power of multi-agent teams of characters.

3 Application of ABL in ***Façade***

We are using ABL to implement the believable agent layer of the *Façade* interactive drama architecture (Fig. 1). *Façade* is a serious attempt to move beyond traditional branching or hyper-linked narrative, to create a dramatically interesting virtual world inhabited by computer-controlled characters, within which the player experiences a story from a first person perspective (Mateas and Stern [19, 22]). The complete, real-time 3D, one-act interactive drama will be available in a free public release at the end of 2003.

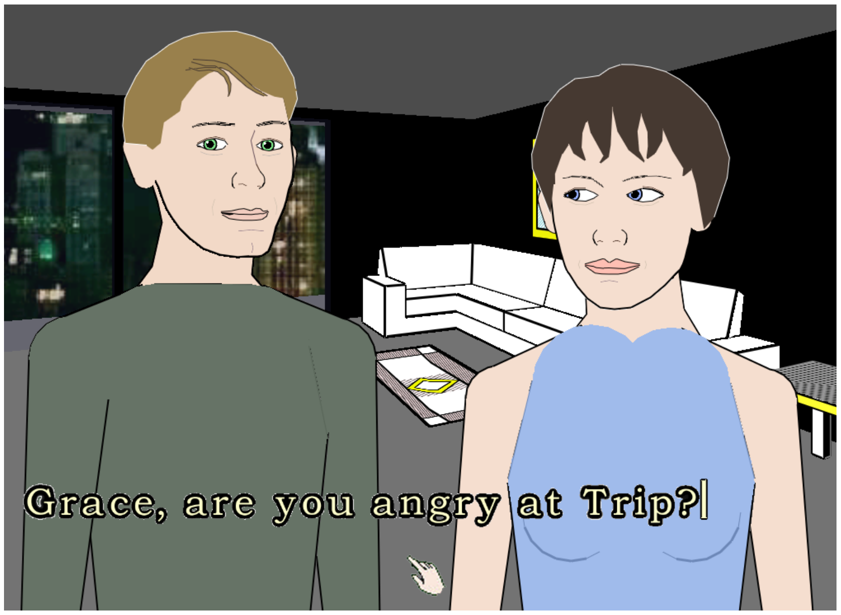

Fig. 1. Screen capture from *Façade*

In the drama, Grace and Trip, a married couple in their early thirties, have invited the player over for drinks. The player soon learns that their marriage is in serious trouble, and in fact, tonight is the night that all their troubles are going to come to the surface. Whether and how their marriage falls apart, and the state of the player's relationship with Grace and Trip at the end of the story, depends on how the player interacts in the world. The player interacts by navigating in the world, manipulating objects, and through natural language dialog.

This project raises a number of interesting AI research issues, including drama management for coordinating plot-level interactivity, broad but shallow support for natural language understanding and discourse management, and autonomous believable agents in the context of interactive story worlds. This chapter focuses on the last issue, describing the idioms developed within ABL for organizing character behaviors. For details of the rest of the *Façade* architecture, including the drama manager and natural language processing system, see (Mateas and Stern [19, 21], Mateas [18]).

4 ABL Overview

This section provides an overview of the ABL language and discusses some of the ways in which ABL modifies or extends Hap. The discussion of joint behaviors, the mechanism for multi-agent coordination, occurs in its own section below.

4.1 Hap

Since ABL re-implements and extends Hap, this section briefly describes the architecture of a Hap agent and the organization and semantics of the Hap language. All examples use the ABL syntax. The definitive reference on Hap is Loyall's dissertation [15].

The ABL compiler is written in Java and targets Java; the generated Java code is supported by the ABL runtime system.

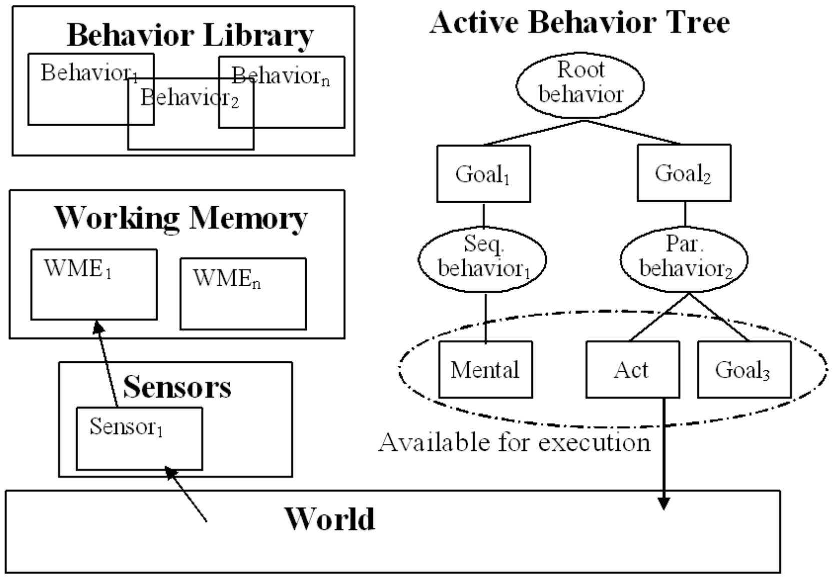

Fig. 2. Architecture of a Hap/ABL agent

The architecture of a Hap/ABL agent appears in Fig. 2. The agent has a library of pre-written behaviors. Each behavior consists of a set of steps, to be executed either sequentially or in parallel, that accomplish a goal. The current execution state of the agent is captured by the active behavior tree (ABT) and working memory. The ABT contains the currently active goals and behaviors. The ABT is a tree rather than a stack because some behaviors execute their steps in parallel, thus introducing parallel lines of expansion in the program state. The leaves of the ABT constitute the conflict set. The agent continuously executes a decision cycle, during which a leaf step is chosen for execution. As each step is executed, it either succeeds or fails. In a sequential behavior, step success makes the next step available for execution. When the last step of a sequential behavior succeeds, or when all steps of a parallel behavior have succeeded, the enclosing behavior succeeds. For both sequential and parallel behaviors, if any step fails, it causes the enclosing behavior to fail. In this way, success and failure propagate through the ABT.

The four basic step types are introduced in the example behavior code below. For now, note that one of the step types is act, which performs a physical action with the agent's body, such as taking a step or grasping an object. The exact details of the execution of a physical action depend on both the agent's body and the world. For example, *Façade* agents have virtual, animated bodies within a real-time, graphical, 3D story world; however, one could just as well use ABL to implement behavior for physical robot agents, text-based agents, etc.

Working memory contains any information the agent needs to keep track of during execution. This information is organized as a collection of working memory elements (WMEs). WMEs are like instances in an object-oriented language; every WME has a type plus some number of typed fields that can take on values. WMEs are also the mechanism by which an agent becomes aware of sensed information. Sensors report information about changes in the world by writing that information into WMEs. Hap/ABL has a number of mechanisms for writing behaviors that are continuously reactive to the contents of working memory, and thus to sensed changes in the world. The details of sensors, like actions, depend on the specific world and agent body.

4.2 Example Behaviors

Hap/ABL programs are organized as collections of behaviors. In the example sequential behavior shown below, an agent waits for someone to knock on a door, sighs, then opens the door and greets the guest.

```
sequential behavior AnswerTheDoor() {
  WME w;
  with (success_test { w = (KnockWME) } ) wait;
  act sigh();
  subgoal OpenDoor();
  subgoal GreetGuest();
  mental_act { deleteWME(w); }
}
```

This behavior demonstrates the four basic step types, namely wait, act, subgoal, and mental_act. wait steps are never chosen for execution; a naked wait step in a sequential behavior would block the behavior from executing past the wait. However, when combined with a success_test, a wait step can be used to make a demon that waits for a condition to become true. Success tests are continuously monitored conditions that, when they become true, cause their associated step to immediately succeed. Though in this example the success_test is associated with a wait step to make a demon, it can be associated with any step type.

Success tests, as well as other tests that will be described shortly, perform their test against the agent's working memory. In this example, the success_test is looking for WMEs of type KnockWME, which presumably is placed in the agent's working memory when someone knocks on a door. Since there are no field constraints in the test, the test succeeds as soon as a KnockWME appears.

An act step tells the agent's body (sensory-motor system) to perform an action. For graphical environments such as *Façade*, physical acts will ultimately be translated into calls to the animation engine, though the details of this translation are hidden from the Hap/ABL program. In this example, the act makes the body sigh. Note that physical acts can fail – if the sensory-motor system determines that it is unable to carry out the action, the corresponding act step fails, causing the enclosing behavior to fail.

Subgoal steps establish goals that must be accomplished in order to accomplish the behavior. The pursuit of a subgoal within a behavior recursively results in the selection of a behavior to accomplish the subgoal.

Mental acts are used to perform bits of pure computation, such as mathematical computations or modifications to working memory. In the final step of the example, the mental_act deletes the KnockWME (making a call to a method defined on ABL agents), since the knocking has now been dealt with. In ABL, mental acts are written in Java.

The next example demonstrates how Hap/ABL selects a behavior to accomplish a subgoal through signature matching and precondition satisfaction.

```
sequential behavior OpenDoor() {
  precondition {
    (KnockWME doorID :: door)
    (PosWME spriteID == door pos :: doorPos)
    (PosWME spriteID == me pos :: myPos)
    (Util.computeDistance(doorPos, myPos) > 100)
  }
  specificity 2;
  // Too far to walk, yell for knocker to come in
  subgoal YellAndWaitForGuestToEnter(doorID);
}

sequential behavior OpenDoor() {
  precondition { (KnockWME doorID :: door) }
  specificity 1;
  // Default behavior - walk to door and open
}
```

In this example there are two sequential behaviors OpenDoor(), either of which could potentially be used to satisfy the goal OpenDoor(). The first behavior opens the door by yelling for the guest to come in and waiting for them to open the door. The second behavior (details elided) opens the door by walking to the door and opening it. When AnswerTheDoor() pursues the subgoal OpenDoor(), Hap/ABL determines, based on signature matching, that there are two behaviors that could possibly open the door. The precondition of both behaviors is executed. In the event that only one of the preconditions is satisfied, that behavior is chosen as the method to use to accomplish the subgoal. In the event that both preconditions are satisfied, the behavior with the highest specificity is chosen. If there are multiple satisfied behaviors with highest specificity, one is chosen at random. In this example, the first OpenDoor() behavior is chosen if the lazy agent is too far from the door to walk there ("too far" is arbitrarily represented as a distance > "100").

The precondition demonstrates the testing of the fields of a WME. The :: operator assigns the value of the named WME field on the left of the operator to the variable on the right.[3] This can be used both to grab values from working memory that are then used in the body of the behavior, and to chain constraints through the WME test.

The last example demonstrates parallel behaviors and context conditions.

```
parallel behavior YellAndWaitForGuestToEnter(int doorID) {
  precondition { (CurrentTimeWME t :: startT)}
  context_condition {(CurrentTimeWME t <= startT + 10000)}
  number_needed_for_success 1;
  with (success_test {{DoorOpenWME door == doorID)}) wait;
  with (persistent) subgoal YellForGuest(doorID);
}
```

In a parallel behavior, the steps are pursued simultaneously. YellAndWaitForGuestToEnter(int) simultaneously yells "come in" towards the door (the door specified by the integer parameter) and waits to actually see the door open. The persistent modifier on the YellForGuest(int) subgoal makes the subgoal be repeatedly pursued, regardless of whether the subgoal succeeds or fails (one would imagine that the behavior that does the yelling always succeeds). The number_needed_for_success annotation (only usable on parallel behaviors) specifies that only one step has to succeed in order for the behavior to succeed. In this case, that one step would be the demon step waiting for the door to actually open. The context_condition is a continuously monitored condition that must remain true during the execution of a behavior. If the context condition fails during execution, then the behavior immediately fails. In this example, the context condition tests the current time, measured in milliseconds, against the time at which the behavior started. If after 10 seconds the door has not

[3] In ABL, a locally-scoped appropriately typed variable is automatically declared if it is assigned to in a WME test and has not been previously explicitly declared.

yet opened (the guest is not coming in), then the context condition will cause the behavior to fail.

As failure propagates upwards through the subgoal chain, it will cause the first OpenDoor() behavior to fail, and eventually reach the OpenDoor() subgoal in AnswerTheDoor(). The subgoal will then note that there is another OpenDoor() behavior that has not been tried yet and whose precondition is satisfied; this behavior will be chosen in an attempt to satisfy the subgoal. So if the guest does not enter when the agent yells for awhile, the agent will then walk over to the door and open it.

These examples give a sense for the Hap semantics which ABL reimplements and extends. There are many other features of Hap implemented in ABL that are not possible to re-describe here, including how multiple lines of expansion mix (based on priority, blocking on physical acts, and a preference for pursing the current line of expansion), declaration of behavior and step conflicts (and the resulting concept of suspended steps and behaviors), and numerous annotations that modify the default semantics of failure and success propagation (Loyall [15]).

4.3 ABL Extensions

ABL extends Hap in a number of ways, including:

- Generalizing the mechanisms for connecting to the sensory-motor system. The ABL runtime provides abstract superclasses for sensors and actions. To connect an ABL program to a new sensory-motor system (e.g., an animation engine, a robot), the author defines specific sensors and actions as concrete subclasses of the abstract sensor and action classes. ABL also includes additional language constructs for binding sensors to WMEs. ABL then takes responsibility for calling the sensors appropriately when bound WMEs are referenced in working memory tests.
- Atomic behaviors. Atomic behaviors prevent other active behaviors from mixing in. Atomic behaviors are useful for atomically updating state (e.g. updating multiple WMEs atomically), though they should be used sparingly, as a time-consuming atomic behavior could impair reactivity.
- Reflection. ABL gives behaviors reflective access to the current state of the ABT, supporting the authoring of meta-behaviors that match on patterns in the ABT and dynamically modify other running behaviors. Supported ABT modifications include succeeding, failing or suspending a goal or behavior, and modifying the annotations of a subgoal step, such as changing the persistence or priority. Safe reflection is provided by wrapping all ABT nodes in special WMEs. Pattern matching on ABT state is then accomplished through normal WME tests. A behavior can only touch the ABT through the reflection API provided on these wrapper WMEs.
- Multiple named memories. Working memories can be given a public name, which then, through the name, are available to all ABL agents. Any WME

test can simultaneously reference multiple memories (the default memory is the agent's private memory). In *Façade*, named memories are useful for giving agents access to a global story memory.

The most significant ABL extension of Hap is support for joint goals and behaviors, described in the following section.

5 Joint Goals and Behaviors

In order to facilitate the coordination of characters in the carrying out of dramatic action, we extended the semantics of Hap, in a manner analogous to the STEAM multi-agent coordination framework (Tambe [31]). This section describes *joint goals and behaviors*, ABL's support for multi-agent coordination.

The driving design goal of joint behaviors is to combine the rich semantics for individual expressive behavior offered by Hap with support for the automatic synchronization of behavior across multiple agents.

In ABL, the basic unit of coordination is the joint goal. When a goal is marked as joint, ABL enforces, in a manner transparent to the programmer, coordinated entry into and exit from the behaviors chosen to accomplish the goal. The keyword `joint` can be used to modify both goals and behaviors, telling ABL that entry into and exit from the joint behavior should be automatically coordinated with team members.

5.1 ABL's Under-the-hood Negotiation Process for Joint Goals and Behaviors

Entry into a behavior occurs when the behavior is chosen to satisfy a subgoal. *Exit* from the behavior occurs when the behavior succeeds, fails, or is suspended. ABL's algorithm for executing a joint subgoal and coordinating entry appears in Fig. 3.

When ABL executes a joint goal, a behavior is chosen for the goal using normal Hap behavior selection methods, with the additional constraint that the behavior must be joint (marked with the `joint` keyword).

Joint behaviors include a specification of the team members who must participate in the behavior. If a joint behavior is found for the joint goal, ABL marks the goal as negotiating and begins negotiating entry with team members specified in the joint behavior. The negotiating joint goal is removed from the conflict set, blocking that line of expansion until negotiation completes. All other parallel lines of expansion are still pursued. If the negotiation takes awhile, perhaps because there are a large number of distributed teammates who are synchronizing during the negotiation, all negotiating agents continue to execute the decision cycle and engage in behavior. An intention-to-enter message is sent to all team members. The initiating message includes information about the goal signature and arguments.

1. The initiating agent chooses a joint behavior for the joint goal based on signature matching, precondition satisfaction, and specificities.
2. If a joint behavior is found for the joint goal, mark the goal as negotiating and broadcast an intention to enter the goal to all team members, otherwise fail the goal.
3. If all team members respond with an intention to enter the joint goal, add the joint behavior (and behavior children) to the ABT.
4. If any team member reports an intention to refuse entry to the joint goal, broadcast an intention to refuse entry and fail the behavior when all team members respond with an intention to refuse entry.

Fig. 3. Agent initiating a joint behavior via joint subgoal execution

The goal remains in the *negotiating* state until all team members respond with an intention to enter or an intention to refuse entry. If all agents respond with intention-to-enter messages, this signals that all agents in the team have found appropriate behaviors in their local behavior libraries; the goal state is changed to *executing*, and the selected behavior and its steps are added to the ABT. If any agent responds with an intention to refuse entry, presumably because, given the goal signature and goal arguments, it could not find a satisfied joint behavior, the initiating agent sends all team members an intention to refuse entry. When all agents report that they intend to refuse entry, the initiating agent fails the joint behavior (whose steps never actually got a chance to execute). This causes the goal to attempt to find a different joint behavior with satisfied precondition, perhaps one with a different set of team members. Just as with a normal (non-joint) goal, if no such alternate behavior can be found, the goal fails.

Figure 3 shows the entry negotiation algorithm for the *initiator* of a joint goal, that is, the agent who originally executes the joint goal step, and who thus begins the joint behavior selection and negotiation process. The teammates of a joint goal initiator use a similar negotiation algorithm. The only difference is that for non-initiators, a joint goal with appropriate signature and arguments must be created and attached to the root collection behavior of the ABT.

1. An initiating agent broadcasts to all team members an intention to exit (either succeed, fail, or suspend) an executing joint goal.
2. All agents receiving an intention to exit respond by broadcasting to all team members their own intention to exit (succeed, fail, or suspend).
3. When all team members respond with the appropriate intention to exit, the joint goal is succeeded, failed or suspended as appropriate.

Fig. 4. Agent exiting a joint behavior

The algorithm for coordinating exit from a joint behavior is shown in Fig. 4. For example, assume that a joint behavior has been successfully entered by a team. At this point each member of the team is executing a joint behavior from their local behavior library with the same signature, arguments, and team members. One of the team members, in executing their local joint behavior, encounters a condition where they should exit the behavior. Perhaps the last step of the behavior succeeds, causing the joint behavior and goal to succeed, or the context condition fails, causing the joint behavior and goal to fail, or a higher priority conflicting goal (either joint or non-joint) enters the ABT, causing the joint goal to suspend. The agent encountering this situation becomes the initiator of the intention to exit (the exit initiator does not have to be the same agent as the entry initiator). The exit initiator marks the joint goal as negotiating and broadcasts the appropriate intention to exit to all team members. While the joint goal is in the negotiating state it blocks that line of expansion; all other lines of expansion in the ABT are still active.

As each team member receives an intention to exit, it marks its local version of the joint goal as negotiating and broadcasts an exit intention. Once exit intentions have been received from all team members, an agent exits the negotiating goal (succeeds, fails or suspends the goal).

5.2 Example of Basic Joint Goal and Behavior Support

This section provides a simple example of the joint goal negotiation protocol in action, based on the follow-the-leader behavior of the *Woggles* [16].

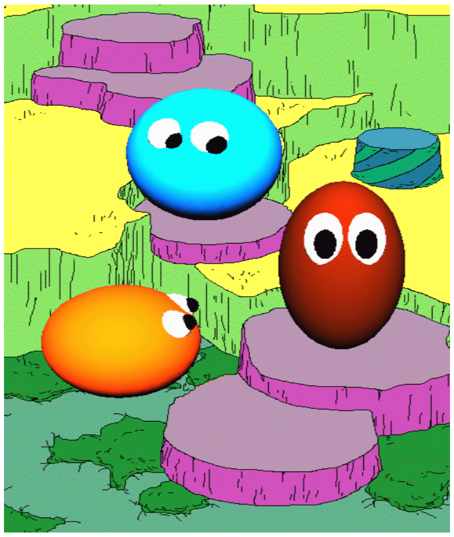

Fig. 5. Close-up of three woggles

The Woggle world, an early demonstration system produced by the Oz Project, consists of a Dr. Seuss-like landscape inhabited by three Woggles – the shy Shrimp, the aggressive Wolf, and the friendly Bear (Fig. 5.2). As the Woggles play, fight and hang out with each other, the player is able to enter the Woggle world as a fourth Woggle.

The diagram in Fig. 6 shows the original behavior structure for a follower playing follow-the-leader. The behavior is decomposed into three subbehaviors, one to copy the leader's jumps, one to copy the leader's squashes, and one to monitor whether the follower is falling too far behind the leader. Each of these behaviors is in turn decomposed into a sensing behavior that gathers information from the world (e.g. see jump, check if you are behind), and a behavior that acts on the sensed information (e.g. copy the jump recorded by the "see jump" behavior). Communication between behaviors takes place via memory elements posted to and matched from the agent's working memory.

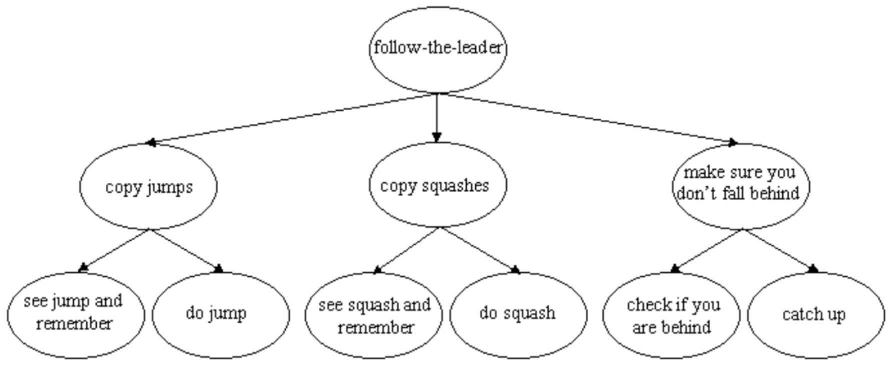

Fig. 6. Original behavior structure for the follower

The diagram in Fig. 5.2 shows the original behavior structure for the leader playing follow-the-leader. The top level behavior is decomposed into two subbehaviors, one to do "fun stuff" (the hopping and squashing that the follower will copy) and one to monitor whether the follower has fallen behind. The "fun stuff" behavior is further decomposed into three different ways to have fun. The "make sure follower does not fall behind" behavior is decomposed into a sensing behavior that monitors the follower's activity, and a behavior that waits for the follower to catch up in the event that the follower did fall behind. Note that both Fig. 6 and Fig. 5.2 elide the sequential structure of the behaviors, showing only the persistent, parallel, subgoal structure. The complete "lead-the-follower" behavior first chooses a co-Woggle to invite, moves over to the invitee, offers an invitation to play follow-the-leader (using Woggle body language), and then, if the invitee signals that the invitation is accepted, starts the two parallel behaviors “fun stuff” and “monitor follower” diagrammed in Fig. 5.2.

For two Woggles to play a game of follow-the-leader, one of the Woggles must first decide that it wants to be a leader and successfully invite the other Woggle to be a follower. The two Woggles then independently execute their respective behavior hierarchies. These two independent hierarchies coordinate via sensing, by mutually monitoring each other’s physical activities. In addition to the follow-the-leader behavior hierarchy, both Woggles have a number of other behaviors executing in parallel. These behaviors are monitoring the world for certain actions, such as someone saying “hi”, a friend being attacked by someone else, someone inviting the Woggle to play a game, etc. If the follower pauses in the middle of the game to respond to one of these world events, perhaps suspending its local follow-the-leader behavior hierarchy, the leader will experience this as the follower falling behind. If the follower takes too long to get back to the game, the leader will “time out” and the lead-the-follower behavior will fail (stop executing with failure). The leader will then start doing something else.

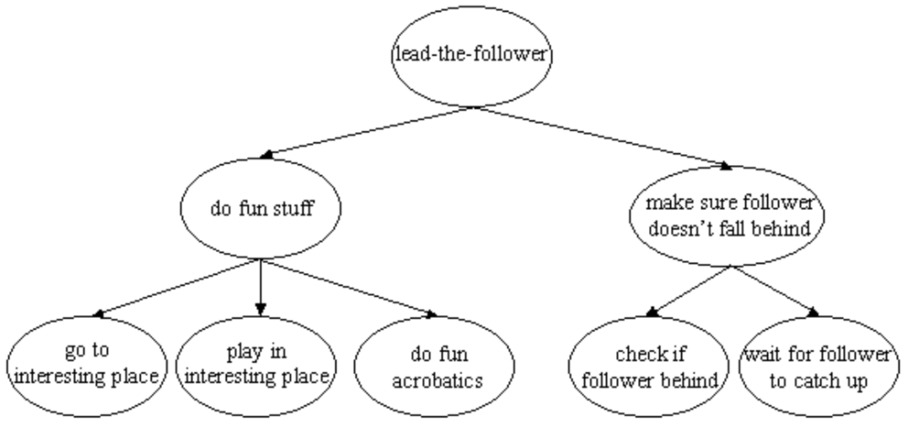

Fig. 7. Original behavior structure for the leader

However, unless similar timeouts have been placed in the right behaviors in the follower, the follower, after completing the interruption, will unsuspend and continue playing follow-the-leader. In fact, the original Woggle code *does not* have the appropriate timeouts in the follower, and so this condition can happen.

At this point, the former leader is jumping around the world doing its own thing while the follower dutifully follows behind copying the leader’s actions; the leader is not aware that the follower’s actions are in any way related to the leader, and the follower has no idea that the leader is no longer playing follow-the-leader. This is one example of the coordination failures that can happen

```
//Leader's version of FollowTheLeader
joint parallel behavior FollowTheLeader {
      teammembers Shrimp, Bear;
      precondition { <I'm a leader and feel like leading> }

      subgoal DoFunStuff();
      subgoal MakeSureFollowerDoesntFallBehind();
}

//Follower's version of FollowTheLeader
joint parallel behavior FollowTheLeader {
      teammembers Shrimp, Bear;
      precondition { <I'm a follower and feel like playing> }

      subgoal CopyJumps();
      subgoal CopySquashes();
      subgoal MakeSureYouDontFallBehind();
}
```

Fig. 8. Joint behaviors for follow-the-leader

even in a rather simple joint action when the joint activity is produced through the ad hoc synchronization of independent behavior hierarchies.

Using ABL's joint behaviors, the top of the leader's and follower's follow-the-leader behavior hierarchies are shown in Fig. 8. To simplify the example, consider just two of the Woggles, Shrimp and Bear. Since either can be a leader or follower, both Woggles have both the leader and follower versions of the behavior in their behavior libraries. One of them, say Bear, decides to play follow-the-leader – this decision is made by logic in some other behavior, perhaps a high level motivational behavior, resulting in the creation of a WME, LeaderWME, indicating that Bear wants to lead a game of follow-the-leader, a body language request to Shrimp to play, and the execution of joint subgoal FollowTheLeader().

The execution of the joint subgoal results in ABL trying to find a satisfied joint behavior to accomplish the goal. The preconditions distinguish between the leader and follower cases. If the behaviors did not have preconditions testing LeaderWME, then the initiator of FollowTheLeader() might inadvertently select the follower version of the behavior. Sensing could also be used to distinguish the two cases, selecting the leader version if a body language request to play from another Woggle has not been recently seen, and the follower version if it has. Once ABL has selected the leader version of joint behavior FollowTheLeader() for Bear, the subgoal FollowTheLeader() is marked as negotiating and a request-to-enter is sent to Shrimp. ABL creates a joint subgoal FollowTheLeader() at the root of Shrimp's ABT and selects the follower version of the behavior from his joint behavior library, again using preconditions to distinguish cases. Note that the preconditions can also be used to add personality-specific tests as to whether the Woggle feels like playing follow-the-leader. In Shrimp's case, for example, Shrimp may only feel like playing if he has not recently been picked on by another Woggle. Assuming that Shrimp's precondition is satisfied, ABL sends an intention-to-enter from Shrimp to Bear. Both Shrimp and Bear have received intentions-to-enter from all team members, so ABL adds their respective selected behaviors to their ABT's; they are now both playing follow-the-leader.

Once coordinated entry into FollowTheLeader() is established, they continue playing follow-the-leader until one of them exits the behavior. Perhaps Wolf threatens Shrimp in the middle of the game, causing Shrimp to engage in high priority fear reaction that suspends his local FollowTheLeader() goal. The goal is marked as negotiating and an intention to suspend is sent to Bear. ABL marks Bear's goal as negotiating and sends an intention to suspend to Shrimp. They have both received intentions to suspend from all team members, so ABL locally suspends the FollowTheLeader() goal for each. Similar exit negotiations ensure synchronization on goal success and failure. Every team member is guaranteed that if it is locally executing the joint goal FollowTheLeader(), all team members are executing the joint goal FollowTheLeader().

5.3 Nested / Multiple Joint Goals and Behaviors

Joint goals and behaviors thus synchronize behavior execution across agents; the entry into a joint behavior is precisely a synchronization point. Joint and individual behaviors can be nested arbitrarily within the behavior hierarchy, depending on the granularity of the synchronization required. In the simple joint behaviors in Fig. 8, only the FollowTheLeader() behavior is joint. However, smaller granularity behaviors could also be joint. For example, jump and squash could be implemented as joint behaviors within the follow-the-leader behavior hierarchy. When a joint jump is entered, it would coordinate the leader and followers for the specific act of jumping. In essence, this would establish an automatic pattern of communication between the Woggles saying "now the leader is going to do a jump and all the followers should copy it". In addition, an agent can be committed to multiple simultaneous joint goals with different team members. For example, Shrimp could be a follower committed to a FollowTheLeader() goal with Bear while simultaneously committed to a Hassle() goal with Wolf (a goal coordinating Wolf and Shrimp in Wolf hassling Shrimp). As the two behaviors mixed together, Shrimp would keep a wary eye on Wolf, occasionally slinking or groveling, while trying to keep up in follow-the-leader with Bear.

5.4 Full Complexity of Joint Goal Negotiation

So far this chapter has described a simplified version of ABL's automatic joint goal negotiation protocol. The full negotiation protocol must appropriately handle a number of edge cases and race conditions, specifically:

1. ABL must achieve a consistent team state when multiple team members simultaneously intend to exit a joint goal for inconsistent reasons (e.g. three different team members simultaneously try to fail, succeed, and suspend the same joint goal).
2. ABL must resolve inconsistencies resulting from out-of-order receipt of negotiation messages. Messages may arrive out of order due to the message transport mechanism or due to differences in the execution speed of different team members.
3. ABL must resolve negotiation conflicts arising within a single agent due to the parallel pursuit of multiple goals. For example, in the middle of a negotiation to enter a goal, a parent goal may suspend, and thus cause the recursive suspension of all individual and joint goals in the tree rooted at the parent goal.

The details of the full joint goal framework are beyond the scope of this chapter; please see [18] for more detail. The bottom line is that joint goals and behaviors guarantee coordinated entry and exit across arbitrarily sized teams consisting of fully asynchronous, arbitrarily scheduled team members.

6 *Façade* ABL Idioms: Implementing Dramatic Beats

Developing a believable agent language such as ABL involves simultaneously defining and implementing language constructs that support the authoring of expressive behavior, and the exploration of idioms for expressive behavior *using* the language. This section describes the ABL idioms developed for *Façade.*

In [22, 20] we argued that much work in believable agents is organized around the principle of strong autonomy, and that, for story-based believable agents, this assumption of strong autonomy is problematic. An agent organized around the notion of strong autonomy chooses its next action based on local perception of its environment plus internal state corresponding to the goals and possibly the emotional state of the agent. All decision making is organized around the accomplishment of the individual, private goals of the agent. But believable agents in a story must also participate in story events, which requires making decisions based on global story state (the entire past history of interaction considered *as* a story) and tightly coordinating the activity of multiple agents so as to accomplish joint story goals. In order to resolve the tension between local and global control of characters, in *Façade* we organize behaviors around the *dramatic beat.* In the theory of dramatic writing, the beat is the smallest unit of dramatic action (McKee [23]). In *Façade,* a beat is a ~60-second-long dramatic interaction between characters such as a brief conflict about a topic, a short psychological headgame, or the revelation of an important secret.

Beat-specific ABL behaviors provide the procedural knowledge necessary to perform an interactive dramatic beat. This section focuses on describing the ABL idioms used for authoring such behaviors. In *Façade*, beat sequencing (selecting the next beat to make active) is handled by the drama manager; see [19, 21, 18] for details of the drama manager.

6.1 Beat Goal and Handler Behaviors

Beats are organized around a collection of *beat goal behaviors* – the dramatic content the beat is designed to communicate to the player through animated performance. Our authoring strategy for handling player interaction within a beat is to specify the "canonical" beat goal behavior logic (i.e., what dramatic performance the author intends the beat to accomplish), as well as a collection of beat-specific *handler behaviors* that modify this default logic in response to player interaction. In order to modify the default logic, the handler behaviors make use of meta-ABL functionality to modify the ABT state.

While handler behaviors can in principle arbitrarily modify the ABT state, most fall into one of two general classes – *mix-in* handler behaviors that add an additional beat goal behavior in the middle of a beat while keeping the rest of the sequencing the same, and *re-sequencing* handler behaviors that more radically reorganize the beat goal sequence. The ability to factor behavior into sequences that achieve longer-term temporal structures, and meta-behaviors that modify these longer-term temporal structures, is a powerful idiom enabled by ABL's support for reflection.

6.2 Inside a Beat Goal Behavior

Within individual beat goals, joint goals and behaviors are used to coordinate the characters in the performance of dramatic action. As can be seen in Fig. 9, the detailed, coordinated, physical performance of even a single line of dialog can be quite rich. The characters must engage in coordinated dramatic performance to accomplish the intentions of the beat goals, while simultaneously pursuing longer-term, individual goals that cross beat goal boundaries. (Cross-beat behaviors are described later in this chapter.) This is achievable because ABL can support the autonomous pursuit of individual goals in parallel with tightly coordinated team activity, in the ways described in the previous section of this chapter.

6.3 Organizing Beat Goal Behaviors

The high-level beat goal specification for an example beat, "Argue about Grace's decorating," appears in Fig. 10. The author intends this beat to reveal conflict in Grace and Trip's marriage by hinting at the fact that there is something weird about Grace's obsessive decorating of their apartment. In this beat Grace is trying to get the player to agree that something is wrong with her current decorating scheme, while Trip is trying to get the player to

Grace: . . . but looking at it here **in** the apartment, it just looks like . . . (laugh half-lighthearted, half-bitter) **a** monstrosity!

Grace physical performance:

- Throughout the line Grace physically adjusts to face the player.
- On the bold words she slightly nods her head and barely raises both arms.
- At the beginning of the line she looks at the armoire with a frown, changing to a serious expression a short way into the dialog.
- At the beginning of the line her mood becomes anxious.

Trip physical performance:

- Throughout the line Trip physically adjusts to face the player.
- At the beginning of Grace's line he turns (just his head and eyes) to look at Grace.
- A short way into Grace's line he gets an impatient expression on his face.
- A short way into Grace's line he reacts with an anxious reaction.

Fig. 9. Detailed performance specification for one line of dialog

Transition In (the no-subtopic version)

G: So, Player, I'm hoping you can help me understand where I went wrong with my new decorating (bitter laugh).
T: (pacifying tone) Oh, Grace, let's not do that.

Address subtopic, part 1 (armoire subtopic)

G: (sigh) You know, when I saw this armoire on the showroom floor, I thought it had such a clean, simple look to it...
T: It's a nice choice.

Address subtopic, part 2 (armoire subtopic)

G: ...but looking at it here in the apartment, it just looks like... (laugh half-lighthearted, half-bitter) a monstrosity!
T: (under-breath impatient sigh) uhh...

Wait Timeout (default version)

(G looks impatiently between the object and the player, T fidgets)

Transition Out (lean towards Grace affinity)

G: Ah, yes, I've been waiting for someone to say that!
T: (frustrated) What are you talking about?
G: Trip, our friend is just being honest about my decorating, which I appreciate.
T: (sigh) But I still think this looks fine... (annoyed)

Fig. 10. Excerpts from the beat goal specification for the "Argue about Grace's decorating" beat

agree that it looks great and everything is fine. This beat is an instance of what we call an “affinity game”.

In our idiom for *Façade,* beats generally have a transition-in beat goal responsible for establishing the beat context and possibly relating the beat to action that happened prior to the beat, a transition-out beat goal communicating the dramatic action (change in values) that occurred in the beat as a function of player interaction within the beat, and a small number of beat goals between the transition-in and transition-out that reveal information and set up the little interaction game within the beat.

In our example decorating beat, the transition-in beat goal introduces the ostensible topic of the beat: Grace thinks something is wrong with her decorating. The particular transition-in shown in Fig. 10 is the most “generic” transition-in, the one to use if the interaction prior to this beat did not have anything to do with the room or decorating. Other transition-in beat goals are available in this beat for cases in which the player has somehow referred to decorating just prior to this beat being sequenced, for example, by referring to an object associated with decorating such as the couch or the armoire[4].

In the body of this beat, the two “address subtopic” beat goals, Grace specifically critiques some aspect of her decorating. Trip objects and thinks it looks fine, revealing conflict between them. For this beat there are a number of different “address subtopic” beat goals for different decorating subtopics, corresponding to different objects in the room; at the beginning of this beat goal behavior, if the player has not referred to a specific object, one is chosen at random, otherwise the object referenced by the player (perhaps in an interaction just prior to this beat) is used.

The beat goal sequence described up to this point is the default logic for the beat, that is, the sequence of activity that would happen in the absence of player interaction. Of course, the whole point of an interactive drama is that the player can interact – thus there needs to be some mechanism for incorporating interaction into the default beat logic. This is the job of *handler behaviors.*

6.4 Incorporating Player Interaction: Handler Behaviors

Each handler behavior is a demon that waits for some particular type of player interaction and “handles” it accordingly. Player interaction includes dialog interaction (in *Façade,* the player can speak to the characters at any time by typing text) and physical interaction (e.g. the player moves and stands next to some object). Every beat specifies some beat-specific handlers; additionally, there are global handlers for handling interactions for which there are no beat-specific responses supplied by the current beat. Handlers are meta-behaviors;

[4] “References” to an object can happen verbally, e.g. the player types “I like the couch”, and/or physically, by standing near an object and looking at it or picking it up.

they make use of reflection to modify the default logic of the beat. Handlers fall into two broad classes: mix-in handlers, which primarily mix a response into the current beat, and re-sequence handlers, which modify the sequencing logic of the current beat.

The most straightforward use of mix-in handlers is to choose a transition-out beat goal when the player interacts during the "wait-timeout" beat goal. The transition-out communicates how the player's interaction within the beat has changed the story situation. One of the primary story values in *Façade* is the player's affinity with the characters, whether the player is allied with Trip, with Grace, or is neutral. Affinity in *Façade* is zero-sum – if the player moves towards positive affinity with Trip, the affinity with Grace becomes negative. For this beat there are three transition-out beat goals, one each for the cases of the affinity changing towards Trip, one for towards Grace, and one for staying neutral. The example transition-out beat goal in Fig. 10 is the one for an affinity change towards Grace; in this case the player has agreed with Grace (e.g. "Yes, the armoire is ugly"), and, perhaps surprisingly, Grace is happy to hear someone say that.

However, the player, free to speak and act at any time, could have interacted earlier in the beat before the body beat goals had completed – that is, before the conflict for the beat had been fully established. For example, imagine that after Grace's line in the transition-in of the decorating beat ("So, Player, I'm hoping you can help me understand where I went wrong with my new decorating"), the player quickly agrees with Grace ("Yeah, it looks bad"). Since the beat conflict has not been established, it would not make sense to choose the transition-out yet. In this case, a beat-specific mix-in beat goal occurs[5], during which Trip says "Well hold on, hold on, take a closer look, give it a chance to soak in a little." To accomplish this, a beat-specific mix-in handler behavior written to handle early-agreement first aborts the current active beat goal behavior, then spawns a particular high-priority beat goal behavior designed to respond to the agreement. Because of its high priority, the newly mixed-in agreement-response beat goal behavior will happen first; then, the original beat goal behaviors will continue in their normal order. Beat goal behaviors are responsible for determining if they had been interrupted by mixed-in beat goals, and are designed to perform their content in alternate ways as needed.

Global mix-in handlers respond to interactions that are not handled within the current local (specific) beat context. For example, imagine that during the decorating beat the player asks if Grace and Trip are planning to ever have children. There is no beat-specific meaning to referring to children within the

[5] The determination that the player's utterance "Yeah, it looks bad" is an agreement with Grace, and further, that agreement in this context should result in a mix-in rather than a transition-out, is handled by *Façade's* natural language processing (NLP) system, not by ABL behaviors. For details on *Façade's* NLP system, see [19, 18].

decorating beat, so a global handler responds to this interaction with a global mix-in.

Finally, consider the case of beat goal re-sequence handlers – handlers that significantly modify the current sequence of beat goals. Within the decorating beat, a re-sequence handler reacts to references to all the objects related to the decorating beat. There are a number of variants of the "address subtopic" beat goals, each of which makes use of a different object to establish the beat conflict. The version of the "address subtopic" beat goals in Fig. 10 uses the armoire; other versions use the couch, the paintings, the wedding picture, etc. If during the beat the player makes a reference, either physically or verbally, to any of these objects, the beat switches subtopics; the switch-subtopic re-sequence handler rewinds the beat goal logic to address the new subtopic.

6.5 Cross-Beat behaviors

In addition to beat goals and handlers, characters in *Façade* also engage in longer-term behaviors that cross beat goal and beat boundaries. The performance of these longer-term behaviors happens in parallel with the performance of beat goals. An example cross-beat behavior is the staging behavior that an agent uses to move to certain dramatically significant positions (e.g. close or far conversation position with the player or another agent, into position to pickup or manipulate an object, etc.). A staging request to move to close conversation position with the player might be initiated by the first beat goal in a beat. The staging goal is spawned to another part of the ABT. After the first beat goal completes its behavior, other beat goals and handlers can happen as the agent continues to walk towards the requested staging point. At any time during a cross-beat behavior, beat goals and handlers can use reflection to find out what cross-beat behaviors are currently happening and succeed or fail them if the cross-beat behaviors are inappropriate for the current beat goal's or handler's situation.

Additional cross-beat behaviors include fixing a drink for the player (moving to the bar, making a drink, walking to the player and handing her the drink, all in parallel with the performance of beat goals) and personality behaviors such as Grace recollecting a dream she had last night, or Trip obsessively consulting his fortune-telling crystal ball toy.

7 Coupled ABL Agents Form A Multi-Mind

When authoring multiple coordinating agents, the common approaches are either to view the multiple agents as the effectors of a single, centralized control system (one mind), or as completely separate entities (many minds) that coordinate via sensing and explicit communication. This is typically treated as an architectural decision, made once up front and then frozen. In contrast, joint goals, by introducing complex patterns of coupling among a collection of

ABL agents, open up a spectrum between one and many minds in which the degree of coupling can dynamically change. Thus, rather than architecturally committing to the one-mind or many-minds extreme, the authors of ABL agents can dynamically tune the degree of coupling as they author behaviors. This variable coupling can be best understood by comparing it in more detail to the one-mind and many-minds approach.

In the one-mind approach, a collection of agents are really the different effectors of a single entity. This single entity controls the detailed, moment-by-moment activity of all the "agents". One can certainly imagine writing such an entity in ABL; sensors and actions would be parameterized to refer to specific "agents". In an interactive drama context, this is similar to the story plans approach employed by Lebowitz [11, 12], in which he generated non-interactive episodic soap operas (as text) by using story plans (as opposed to character plans) to coordinate multiple characters in specific story events. One-mind provides maximum coordination control, but also introduces maximum program complexity. Besides the usual data hiding and modularity arguments that such a program would be hard to write and understand, and that, consequently, unforeseen side effects would arise from cross-talk between "agents", there is the additional issue that much of the combinatorics of agent interaction would be thrust upon the author. All simultaneous agent activity, whether explicitly coordinating or not, has to be explicitly authored.

The many-minds approach is the intuitive model of strong autonomy. Agents individually pursue their own goals and behaviors. Any coordinated activity arises from sensed coordination between agents. The internal details of agents are hidden from each other, providing the data hiding and modularity that makes programs easier to write and understand. Agent interaction is mediated by the world; much of the combinatorics of agent interaction arises through the world mediation without having to be explicitly authored. But, dramatic interactions in a story world require a degree of coordination difficult to achieve with sensing or ad hoc communication mechanisms [22, 20].

Joint goals, by introducing complex patterns of coupling between teams of ABL agents, open up a middle ground in this apparent dichotomy between one and many minds. When an ABL agent participates in a joint goal, the execution details of its ABT now depend on *both* its autonomous response to the environment as it pursues its individual goals and behaviors, *and* on the execution details of its team members' ABTs, but only to the degree those execution details impinge on the joint goal. With joint goals, a collection of agents becomes a variably coupled multi-mind, neither a single master entity controlling a collection of puppets, nor a collection of completely autonomous agents, but rather a coupled system in which a collection of ABTs influence each other, not arbitrarily, but in a manner controlled by the semantics of joint goal commitment. At any point in time, an ABL agent may hold multiple simultaneous joint goals, potentially with different teams. These joint goals fully participate in the rich, cascading effects of normal ABT nodes; only now the web of communication established between specific nodes in multiple

ABTs allows ABT execution effects to cascade *across* agents as well as *within* agents. As the number and frequency of joint goal commitments across a collection of agents increases, the collection of agents is moving towards a one-mind. As the number and frequency decreases, the collection of agents is moving towards many minds.

8 Related Work

As mentioned previously, ABL builds on the Oz Project work on believable agents (Bates, Loyall, and Reilly [2], Neal Reilly [25], Loyall [15], Sengers [28]), both technically, in that ABL is a re-implementation of Hap adding additional features and language constructs, and philosophically, in that ABL is informed by the Oz stance on believability.

The Media Lab's Synthetic Character group explores architectures that are based on natural, animal systems, particularly motivated by the ethological study of animal behavior (Blumberg [3]). Their recent architecture, C4 (Burke et.al. [4]), builds on their previous architectures, and includes a focus on learning, particularly reinforcement learning for action selection (see Yoon, Blumberg, and Schneider [32]) for a discussion of animal training techniques applied to believable characters). Their work is grounded in the premise that modeling realistic, animal-like, sensory and decision making processes is necessary to achieve believability, particularly the appearance of self-motivation and the illusion of life.

The Virtual Theater Project at Stanford has explored the use of explicitly represented character models in synthetic actors. For example, in the *Master and Servant* scenario, the agents make explicit use of the notion of *status*, borrowed from improvisational acting, to condition their detailed performance of a dramatic scenario (Hayes-Roth, van Gent, and Huber [10]). In the *Cybercafe*, the agents make use of explicit personality traits (e.g. confidence, friendliness), borrowed from trait theories in psychology, to condition the selection and performance of behaviors (Rousseau and Hayes-Roth [27]). More recent work has focused on building annotated environments in which a character dynamically gains new competencies and behaviors from objects in the environment (Doyle [6], Doyle and Hayes-Roth [7]). In this approach, a character's core, invariable features are factored out from the character's environment-specific capabilities and knowledge, with the later being represented in the environment rather than in the character.

A number of groups have explored the use of believable agents in educational simulations. Such work requires that the agent simultaneously communicate its personality while achieving pedagogical goals. The IntelliMedia Project at North Carolina State University has used animated pedagogical agents to provide advice to students in constructivist learning environments (Lester et. al. [14], Lester and Stone [13]). The group has also performed studies to determine whether using agents to deliver advice in such environments actually improves student learning vs. providing the same advice in a non-agent-based form (Moreno, Mayer and Lester [24]). The Institute for Creative Technologies at USC is building story-based military training environments inhabited by believable agents. The agent architecture makes use of a cognitive appraisal model of emotion (Gratch and Marsella [9]) built on top of the STEVE agent architecture (Rickel and Johnson [26]).

Cavazza, Charles and Mead [5] are exploring the use of deliberative character planning for interactive story. They use hierarchical task-network planning to generate plans for character goals. Plots are generated as a function of the interaction between character plans. When the steps of a plan fail (perhaps because of player interaction), the system replans in the new situation.

Over the last several years, game designers have built a number of innovative believable agent architectures for use in commercial games. The virtual pets products *Petz* and *Babyz* (Stern, Frank, and Resner [30], Stern [29]) employ a multi-layer architecture in which state machines perform low-level action sequencing while a goal-processing layer maintains higher-level goal state, activating and deactivating state machines as needed. *Creatures* employs an artificial life architecture in which a neural net selects actions, an artificial hormone system modulates the activity of the neural net, and a genome encodes parameters of both the neural net and hormonal system, allowing traits to be inherited by progeny (Grand [8]). Finally, the creatures in *Black & White* make use of decision tree learning to learn to perform actions on the player's behalf, while the simulated people in the *Sims* hill-climb on a desire satisfaction landscape defined by both the internal drives of the agent and the current objects in the environment (objects provide actions for satisfying drives).

9 Conclusion

During the production of Façade, we have written over 100,000 lines of ABL code, constituting the behaviors for dozens of beats, as well as lower-level behaviors (e.g. gesturing) and cross-beat behaviors (e.g. fixing a drink). Based on this experience, we have found ABL to be a robust, powerful programming framework for authoring believable agents. Two of ABL's new features, joint goals and reflection, have proven particularly useful for authoring *story-based* characters. The behavior of story-based characters must, in addition to maintaining reactive, moment-by-moment believability, must also tightly coordinate behavior to carry out dramatic action and engage in long term discourse sequences.

Joint goals, when combined with the rest of ABL's reactive planning framework, enable coordinated dramatic performance while still supporting rich, real-time interactivity. In *Façade*, joint goals coordinate the performance of individual lines of dialog within beat goals. The automatic coordination protocol associated with joint goals ensures that each character "knows" where they are in the performance of a beat goal. As shown in the Woggle example, the ad-hoc sensor-based coordination of even relatively simple and short lived coordinated behaviors is prone to errors. In an experience like *Façade*, in which multiple characters coordinate every 5 to 10 seconds across a heterogeneous collection of behaviors for minutes on end, architectural support for coordination becomes a necessity. But, with ABL's joint goal mechanism, language support for coordination does not compromise the reactivity of an agent's individual decision making. For example, within *Façade's* joint behaviors, the characters continue to respond to contingencies, such as adjusting their position in response to player movement, changing the gestures accompanying a line of dialog to accommodate holding an object, modifying their facial expressions and body language as a function of emotional state, as well as mixing in longer-term behaviors, such as making a drink, all while coordinating on the performance of lines of dialog.

The reflection framework in ABL supports the authoring of meta-behaviors, that is, behaviors that modify the runtime state of other behaviors. In *Façade*, meta-behaviors enabled the beat goals plus handlers idiom described earlier in this chapter, providing a nice solution to the problem of providing longer-term responsive sequential activity, in this case discourse sequences, within a reactive planning framework. Without reflection, one might accomplish such longer-term sequences in two ways, by unwinding the possible temporal sequences in a hierarchy of goals and behaviors, or through a set of flat (non-hierarchically organized) behaviors that references a declarative discourse state. The first case has the positive feature that the longer-term discourse state is represented by the evolving structure of the ABT, but at the expense of increased authorial complexity in both the behavior hierarchy and the preconditions The second case involves a simpler behavior structure, but pays for it with the necessity to maintain declarative state in addition to the execution state of the agent, introducing the usual problems of keeping declarative representations of a dynamic situation up-to-date. ABL's reflection mechanisms enables a third approach, in which the longer-term sequential activity is explicitly represented in the ABT (the canonical sequence), with meta-behaviors modifying the canonical sequence in response to interaction. With meta-behaviors, an ABL agent can engage in a form of look-ahead planning, in which future sequences of activity are constructed by manipulating the reactive execution state (the ABT).

ABL thus provides support for both the coordinated, more deliberative activity required for storytelling, with the moment-by-moment reactivity required for lifelike behavior responsive to player interaction.

References

1. Bates, J.: Virtual reality, art, and entertainment. *Presence: The Journal of Teleoperators and Virtual Environments* **1**(1):133–138 (1992)
2. Bates, J., Loyall, A. B., Reilly, W. S.: Integrating reactivity, goals, and emotion in a broad agent. In: *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, Indiana (1992)
3. Blumberg, B.: *Old Tricks, New Dogs: Ethology and Interactive Creatures.* (Ph.D. Dissertation, MIT Media Lab 1996)
4. Burke, R., Isla, D., Downie, M., Ivanov, Y., Blumberg, B.: CreatureSmarts: The Art and Architecture of a Virtual Brain. In: *Proceedings of the Game Developers Conference*, San Jose, CA (2001) pp 147–166
5. Cavazza, M., Charles, F., Mead, S.: Characters in search of an author: AI-based virtual storytelling. In: *Proceedings of the International Conference on Virtual Storytelling*, Avignon, France (2001)
6. Doyle, P.: Believability through context: Using "knowledge in the world" to create intelligent characters. In: *Proceedings of the International Joint Conference on Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy (ACM Press 2002) pp 342–349
7. Doyle, P., Hayes-Roth, B.: Agents in annotated worlds. In: *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN (1998)
8. Grand, S.: *Creation: Life and How to Make It* (Harvard University Press Cambridge MA 2001)
9. Gratch, J., Marsella, S.: Tears and Fears: Modeling emotions and emotional behaviors in synthetic agents. In: *Proceedings of the 5th International Conference on Autonomous Agents*, Montreal, Canada (ACM Press 2001)
10. Hayes-Roth, B., van Gent, R. Huber, D.: Acting in character. In: *Creating Personalities for Synthetic Actors*, ed. by Trappl, R., Petta, P. (Springer Berlin New York 1997)
11. Lebowitz, M.: Story telling as planning and learning. *Poetics* **14**:483–502 (1985)
12. Lebowitz, M.: Creating characters in a story-telling universe. *Poetics* **13**:171–194 (1984)
13. Lester, J., Stone, B.: Increasing believability in animated pedagogical agents. In: *Proceedings of the First International Conference on Autonomous Agents.* Marina del Rey, CA (1997) pp 16–21
14. Lester, J., Voerman, J., Towns, S., Callaway, C.: Deictic believability: Coordinating gesture, locomotion, and speech in lifelike pedagogical agents. *Applied Artificial Intelligence* **13**(4-5):383–414 (1999)
15. Loyall, A.B.: *Believable Agents.* Ph.D. thesis, Tech report CMU-CS-97-123 (Carnegie Mellon University 1997)
16. Loyall, A.B., Bates, J.: Real-time control of animated broad agents. In: *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO (1993)
17. Loyall, A.B., Bates, J.: Hap: A reactive, adaptive architecture for agents. Technical Report CMU-CS-91-147. Department of Computer Science (Carnegie Mellon University 1991)
18. Mateas, M.: *Interactive Drama, Art and Artificial Intelligence.* Ph.D. thesis, Tech report CMU-CS-02-206 (Carnegie Mellon University 2002)

19. Mateas, M., Stern, A.: Integrating plot, character and natural language processing in the interactive drama Façade. In: *Proceedings of Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2003)*, Darmstadt, Germany (2003)
20. Mateas, M., Stern, A.: Towards integrating plot and character for interactive drama. In: *Socially Intelligent Agents: The Human in the Loop*, ed. by Dautenhahn, K. (Kluwer 2002)
21. Mateas, M., Stern, A.: *Architecture, Authorial Idioms and Early Observations of the Interactive Drama Façade.* Tech report CMU-CS-02-198 (Carnegie Mellon University 2002)
22. Mateas, M., Stern, A.: Towards integrating plot and character for interactive drama. In: *Working notes of the Social Intelligent Agents: The Human in the Loop Symposium*, AAAI Fall Symposium Series. (AAAI Press Menlo Park CA 2000)
23. McKee, R.: *Story: Substance, Structure, Style, and the Principles of Screenwriting* (HarperCollins New York 1997)
24. Moreno, R., Mayer, R., Lester, J.: Life-Like pedagogical agents in constructivist multimedia environments: Cognitive consequences of their interaction. In: *Proceedings of the World Conference on Educational Multimedia, Hypermedia, and Telecommunications (ED-MEDIA)*, Montreal (2000) pp 741–746
25. Neal Reilly, W. S.: *Believable Social and Emotional Agents* Ph.D. Dissertation., School of Computer Science (Carnegie Mellon University 1996)
26. Rickel, J., Johnson, L.: Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *Applied Artificial Intelligence* **13**:343–382 (1998)
27. Rousseau, D., Hayes-Roth, B.: A social-Psychological model for synthetic actors. In: *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN (1998)
28. Sengers, P.: *Anti-Boxology: Agent Design in Cultural Context.* Ph.D. Dissertation. School of Computer Science (Carnegie Mellon University 1998)
29. Stern, A.: Virtual Babyz: Believable agents with narrative intelligence. In: *Working Notes of the 1999 AAAI Spring Symposium on Narrative Intelligence*, ed. by Mateas, M., Sengers, P. (AAAI Press 1999)
30. Stern, A., Frank, A., Resner, B.: Virtual Petz: A hybrid approach to creating autonomous, lifelike Dogz and Catz. In: *Proceedings of the Second International Conference on Autonomous Agents* (AAAI Press Menlo Park CA 1998) pp 334–335
31. Tambe, M.: Towards flexible teamwork. *Journal of Artificial Intelligence Research* **7**:83–124 (1997)
32. Yoon, S.Y., Blumberg, B., Schneider, G.: Motivation driven learning for interactive synthetic characters. In: *Proceedings of Autonomous Agents 2000* (2000)