Action!: A System For Creating and Delivering Multi-Participant Interactive Cinematic Dramas

Georg Gerber, Massachusetts Institute of Technology[1]

Abstract— We describe an alternative form of computer game, which we call a "multi-participant interactive cinematic drama" and present *Action!* an experimental integrated system for creating and delivering such experiences over the Internet. The system provides a set of tools for building virtual dramatic worlds that are presented to human players by artificial agents, which interact in a layered architecture. The agents are programmed using a special purpose object-oriented language that we call the Interactive Distributed Drama Language (IDDL.)

The artificial agents encode dramatic and cinematic rules, which can be modified by human authors to convey artistic qualities specific to a given experience (e.g., to create a suspenseful style for a murder mystery game.) Agents in the *Action!* architecture carry out two processes: *cinematic* and *literary rendering*. In the case *of cinematic rendering*, agents create a series of comic book-like panels by controlling parameters such as the position/angle of a virtual 3D camera, the size and shape of panels, and the placement of panels on the page. With *literary rendering*, agents generate English language descriptions of dramatic events and the inner states of virtual actors (i.e., character inner-voice.) In both processes, simulated emotional models of the virtual actors influence the choices of the director agent. Thus, if an actor is frightened, the director agent will attempt to create images and sentences that artistically convey a sense of fear.

The system also includes a user interface that allows players to control virtual actors using high-level commands to make characters perform physical actions, speak, or alter their facial expressions. Using the system, we implemented a prototype two-person interactive drama called *Dry Bones* that may be played over the Internet. The drama is set in the jungles of Colombia, and participants may play either Marie, a young American dentist in the Peace Corps or Carlos, a desperate wounded soldier. The overall effect is to create a graphic novel-like experience with rich graphics and narrative descriptions that unfold unpredictably.

I. Introduction

Computer entertainment experiences have largely fallen into two categories: dynamic action/strategy oriented games that are highly interactive but repetitive, or branching "stories" that may have rich graphics and engaging scenarios, but are essentially static. In this paper, we present a somewhat different concept for future computer games that we call "multi-participant interactive cinematic drama." Some key principles that define this concept include:

- Interactions are dramatized *on-the-fly* through an artificial intelligence architecture that applies artistic principles from cinema and other media. Moviemaking relies extensively on altering the "mood" of the experience by varying the camera position and angle [1], [2]. Comic books and graphic novels also have dramatic conventions, such as using a particular shape of "panel" or style of layout for emphasis [3], [4]. An interactive experience must be guided by some artificial dramatic intelligence that encodes these conventions and knows how to apply them.

- Participants are guided by a strong authorial/ artistic point-of-view. A completely freeform, unguided experience (such as a chat room) often becomes dull and is rarely dramatic. Thus, in our view a key feature of interactive drama is the ability to allow an "author" to instill her artistry into the experience and create a guiding, but not overly restrictive framework.

- Interactions among players are the focus. Although fully synthetic characters are becoming increasingly sophisticated, we believe that the most interesting interactions will always be among real people. Thus, we view the computer less as a participant than as a facilitator that continually dramatizes interactions among humans. Additionally, since people take on the roles of characters in the drama, the artificial dramatic intelligence should try to help players "stay in character" by advising how a character might act or think in a certain situation.

The experimental *Action!* architecture attempts to address each of the above points and provide a framework for experimenting with this new entertainment medium.

II. Related Work

Several researchers have worked on projects involving aspects of "interactive drama." The Oz Project led by Joseph Bates focused on creating "interactive fiction" worlds populated by "broad and shallow" agents with a

[1] Laboratory for Computer Science, 200 Technology Square #440, Cambridge, MA 02143, georg@mit.edu.

variety of capabilities [5]. Ken Perlin and Athomas Goldberg created the IMPROV system, designed for scripting synthetic actors that exhibit natural motions through high quality virtual "bodies" [6]. Rousseau and Hayes-Roth also described an architecture for creating autonomous actors with various degrees of "directability" [7]. While these works share certain philosophical similarities with ours, they really address different problems such as creating autonomous synthetic characters or virtual 3D actors that move realistically and can be easily "puppeteered." Our system focuses much more on the computer's facilitating and guiding interactions among multiple human players.

Several researchers have also worked on problems in automated presentation and cinematography, such as He *et al's Virtual Cinematographer* [8], which was applied to filming a conversation in an online chat scenario and Kurlander *et al's Comic Chat* system [9], which automatically presents a chat room experience as a simple 2D comic book-like layout. In terms of graphical presentation, these systems share conceptual similarities with *Action!* However, our work extends these ideas by allowing players to interact in a 3D world that is visualized using unlimited camera angles, embedded cinematic rules, local and global aesthetics applied to panel creation, and arbitrary panel shapes. Further, our architecture has numerous additional layers to deal with actor emotional/ cognitive modeling, automated dramatic language output, etc.

III. SYSTEM ARCHITECTURE

The system has three main components: the client, the communications server, and the render server. A client process resides locally on each player's machine and runs the Interactive Distributed Drama Language (IDDL) interpreter (described below). The client process connects to the communications server and runs all the logic relating to a particular virtual actor experience. The communications server's function is essentially to segregate users into "playgroups" (a set number of players in a drama) and broadcast messages within these groups. This server does not maintain the state of the virtual world or run any of the logic pertaining to it. The render server generates images when the client sends commands requesting that a particular scene be rendered.

In the prototype implementation, each client session maintains an identical local copy of the simulated physical world. In addition, each client session simulates unique virtual actor and director agents. Thus, *Action!* separates the representations of physical objects in the virtual world from mental representations. Any changes to physical objects in the virtual world can only come about through special objects we call *effectors*. The communications server ensures that all *effectors* are received in the same order by all members of the play-group. Hence, the physical representations of the world are synchronized in a lock-step fashion (but mental representations can differ.)

IDDL is an interpreted object-oriented language with a number of specialized features, designed to facilitate the rapid development of interactive dramas. While the language has a syntax similar to C++ or Java, it also allows programmers to use expert-system like rules and other features commonly found in "AI languages." Further, any IDDL object can be sent over the network to another IDDL client. The language also implements a wide variety of special purpose features for facilitating vector operations, non-linear optimization, network communications, and text manipulation. It should be noted that IDDL was not designed as an authoring tool for non-programmers — this is an important area for future work.

IV. AI AGENT ARCHITECTURE

A. Overview

Action! uses a layered artificial intelligence architecture that distributes tasks among several agents, with these agents communicating in a semi-hierarchical manner. Figure 1 shows the information flow among agents. Essentially, the actor cognitive agent perceives the virtual physical world and forms plans and goals to take either actions or to alter the actor's internal cognitive state. In doing so, the actor cognitive agent may perceive and remember dramatic episodes (e.g., another actor is perceived as performing an act of kindness.) The actor emotional agent does not perceive the virtual world directly, but rather accesses the actor cognitive agent's dramatic episodes to alter the actor's internal emotional states.

The director agent functions somewhat like a real-world movie director, making high-level decisions about how best to present the drama (both with pictures and text.) As with the actor cognitive agent, the director agent can perceive the virtual physical world. However, the director agent can also sample the actor's cognitive and emotional states. This information is used to form goals and plans for how best to depict the unfolding drama. These plans result in high-level suggestions for several "good" shots (e.g., a medium shot of an actor from various angles.) These suggestions are then communicated to the editor agent. This agent is concerned with global issues in the dramatic presentation, such as not repeating shots on the same page that are visually similar, or trying to create an overall mood. This agent is itself influenced by the director agent and the graphic designer agent. The editor agent picks a best shot and communicates this to the cinematographer agent.

The cinematographer agent is concerned with actually placing the camera (i.e., realizing the high-level shot suggested by the director and picked by the editor agent.)

Further, the cinematographer agent receives specifications of the dimensions of the target image from the graphic designer agent. The cinematographer agent will then attempt to compute the position of the camera to produce this target image while avoiding occlusions, and may crop the image to create a good composition within the frame. If this agent cannot avoid serious occlusions in the given shot (and after using techniques to try to "repair" the shot), the agent will inform the editor agent that this shot has failed. The editor agent will then choose the next best shot suggested by the director agent, and the cinematographer agent will try again. Once a valid shot is found, the cinematographer agent will send the camera and scene parameters to the rendering server, and the actual image will be created.

We call this process of visually depicting the unfolding drama *cinematic rendering*. Correspondingly, we call the process of generating text to describe the drama *literary rendering*. There are two agents primarily responsible for literary rendering: the dramatic writer agent and the inner-voice agent. The dramatic writer agent creates descriptions of physical actions and objects in the virtual world (e.g., one actor walking over to another.) Just as the director agent provides high-level suggestions for visually depicting the scene that inevitably are acted upon by the cinematographer agent, so too does the director provide high-level suggestions to the dramatic writer agent. In the same sense that the cinematographer agent encodes low-level logic for actually determining camera parameters, the dramatic writer agent contains rules for how to combine words and sentence fragments to create the actual descriptions. The dramatic writer agent can access the actor emotional model, so that descriptions are modulated according to the actor's emotional state. The second agent responsible for literary rendering is the inner-voice agent. This agent's role is to describe the actor's inner thoughts and feelings. It is only directly influenced by the actor emotional agent. As with the dramatic writer agent, the inner-voice agent uses rules to combine predetermined words and sentence fragments. The director agent can indirectly influence the inner-voice agent, since it can cause the actor cognitive agent to form dramatic episodes. In this manner, the director agent can use the actor inner-voice to help "coach" the player to stay in character (e.g., make a heroic character have internal feelings of guilt when she acts dishonorably.)

B. Agent plan/goal system

The actor cognitive agent and the director agent use a plan/goal system, an architecture inspired by the work of Schank [10] and Bates *et al* [5]. The basic system assumes that an agent begins with a set of goals. Each goal object has a *success-test*, which evaluates to true if the goal's objective has been obtained. The goal object contains a set of possible plan objects that it can instantiate. A plan consists of a set of goal objects, and a *context-condition* test. Again, this is a test that evaluates to true if applicable conditions are met for the plan to continue. Every goal and plan can have a dynamically calculated priority. A goal will pick the highest priority plan that has a true context condition; a plan will pick the highest priority goals to pursue first. Finally, there are *actions*, which are essentially simple goals that carry out some primitive end-action (they contain no plans.) Goals can also have a conflict set number assigned to them. This is used to deal with goal/action contentions: if multiple goals/actions are possible, only the highest priority goal from each conflict set can be active at one time.

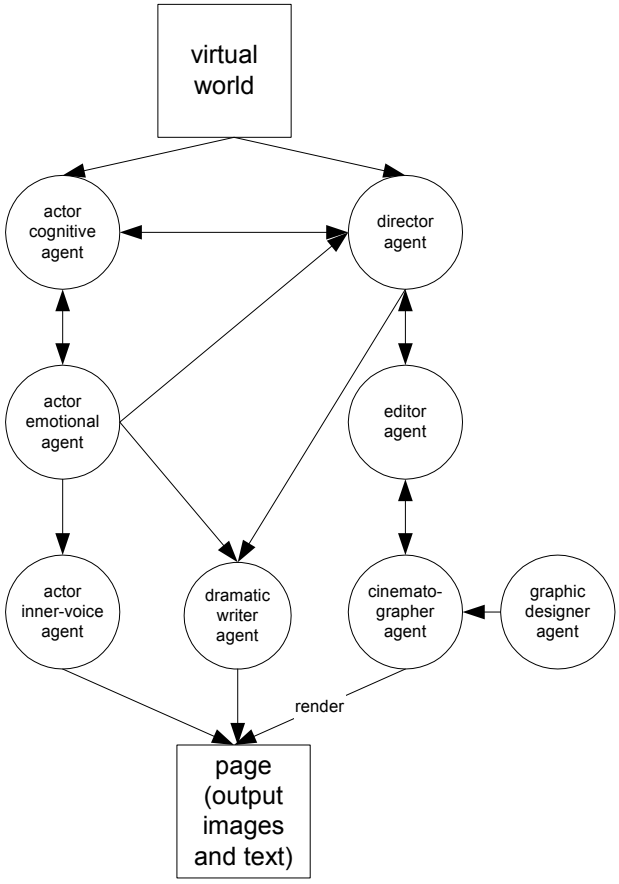

Figure 1: overview of the agent architecture.

V. INTERFACE

The *Action!* interface has four main components:

- *The stage*: the player's primary window on the unfolding physical dramatic action. In the prototype implementation, this area consists of a frame in the browser filled with a series of rectangular panels of different sizes, along with blocks of text that comprise player dialog and generated descriptions.

- *The face*: allows players to generate dialog or change the facial expressions of their virtual actor. In the prototype implementation, players can enter free-form text in a pop-up window. This same window also has a horizontal bar that depicts a spectrum of facial expressions.

- *The body*: allows the player to control the virtual actors' physical actions. In the prototype implementation, this interface consists of a frame in the browser with small renderings of currently visible objects in the environment that will allow the player to act on them (see next section for a further description of player goals.)

- The mind: reveals the actor's inner thoughts to the player. In the prototype implementation, this consists of the top-most frame in the browser, which displays text and small images that reflect actor "thoughts."

VI. Player Goals/Actions

One of the goals of the *Action!* interface is to provide players with more abstract, higher level control of virtual actors; we didn't want this control in itself to be a puzzle-solving exercise. Instead, the player specifies goals to the actor, which in the prototype are relatively simple tasks such as picking up objects, kissing the other player, or sitting down in a chair. However, even these simple tasks may comprise plans containing multiple sub-goals. For instance, the goal of one actor hitting an object instantiates a sequential plan that involves a sub-goal of moving close enough to the object, and another sub-goal for actually hitting the object (see Figure 2.) These sub-goals have the additional advantage that they're reusable and give the actor a degree of "intelligence."

An additional feature of this interface is that possible goals are automatically generated. In this way, the player is presented with the different possible actions her actor can perform. Each object that can be manipulated as a direct by-product of the completion of a player goal can act as a *handle*. As an example, when the player clicks on the icon of the actor Carlos, all possible goals (with appropriate bindings) are returned in which Carlos is either the direct or indirect object. So, a particular instantiated goal might specify the goal for applying a force to Carlos with a hammer. The emotional state of the actor can also modify which goals are available or the plans within the goals. For instance, when an angry actor queries the *ApplyForce* goal, the goal may return a goal to hit another actor. When the actor's mood is more neutral, the apply force goal becomes a handshake; when the mood is passionate, the goal becomes a kiss or embrace.

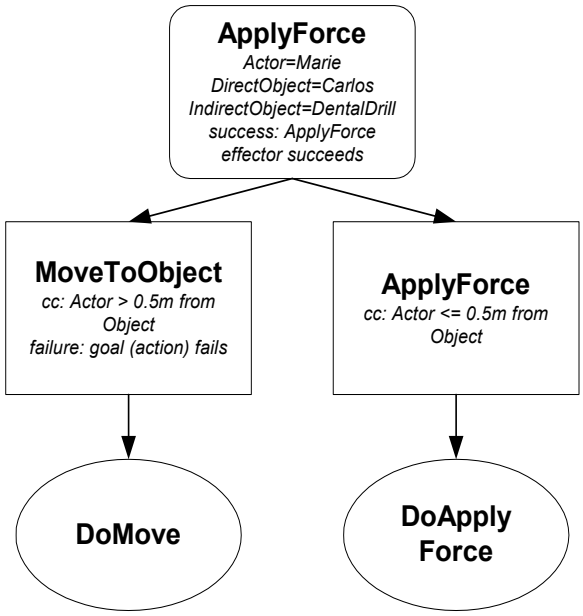

Figure 2: sample player goal/plans (rounded rectangle denotes a goal, rectangles denote plans and ellipses denote actions.)

VII. Emotions and Episodes

Emotions in *Action!* represent the inner affective states of virtual actors. These synthetic emotions are used to create "thoughts" that reveal the actor's inner state as well as to influence *cinematic* and *literary rendering*. In the prototype, we implemented emotions such as gratitude, pity, love, physical distress, fear, remorse, and anger. Note that we use some of the emotions from the models developed by Ortony *et al* [11] and Reilly [12], but also include for convenience physical states that are not "true" emotions.

The actor emotion agent contains instances of *Emotion* objects, which may be linked to one or more *Episodes*. An *Episode* object represents some event that has happened, or is anticipated (e.g., a character hitting another or a memory of atrocities committed by the opposing political party.) The *Episode* object has an *Intensity* slot, and a *Decay* method, which is called at regular intervals and can reduce the episode's intensity value. Using equations suggested in [12], the overall intensity of an *Emotion* object is computed according to:

$$Intensity = \log_2(2^{i_1} + 2^{i_2} + ... + 2^{i_n} + 2^{EM})$$

Here, the i_j represent intensity values for n supporting episodes. The last term is optional, and is only used for some emotions. Here, E represents the intensity of a particular actor expression, and M is a multiplying factor. In this way, the user's input (setting the value of an *Expression* object) can modulate an emotion's intensity. The logarithmic sum function has the effect of neither

allowing several small supporting intensities to combine and cause a large emotional intensity, nor overly damping large intensity values.

The actor cognitive agent has so-called emotional goals. These are essentially goals with plans for recognizing various events, creating episodes, and adding them to the appropriate emotions. For instance, in our prototype we created a plan that detects if one actor gives another a piece of fruit. Since the first actor is initially starving, the action of giving a piece of fruit causes an episode to be created and linked to support a physical satisfaction emotion and a gratitude emotion.

VIII. Cinematic Rendering

A. Director agent

The cinematic rendering process involves high level decisions about what things to depict, then a series of increasingly lower level decisions about how to depict these things in the most aesthetically appropriate ways. At the top level of this architecture is the director agent, which has goals that contain plans that attempt to recognize dramatic situations and suggest cinematic patterns of shots to best depict the situations. For instance, when one player slaps another, the director has a goal that detects this action, and then suggests a sequence of shots for capturing the impact and the player's reaction from a specific point of view.

The director agent builds dynamically prioritized lists of possible shots (priorities are modulated by the editor agent and influenced by parameters from the graphic designer agent.) The highest ranking shot is selected and run; if the shot fails, this process repeats until the list is exhausted. In practice, we always include a special close-up shot in this list (which can never fail.)

B. Cinematographer agent

At a low-level, a shot describes a particular set of camera parameters; at a higher level it conveys some dramatic intent. In *Action!*, an instance of a *Shot* object can be thought of as a cinematographer agent that knows how to film a particular type of shot. The role of the cinematographer agent is to encode the low-level logic necessary to determine the parameters to pass to the 3D renderer. Thus, this agent is concerned with actually calculating the virtual camera's 3D orientation, handling occlusions, and determining how to crop images.

This architecture allows for the specification of high-level constraints for shots within the 2D camera frame (e.g., try to place the actor's face in the center of the left third of the frame.) The cinematographer agent then uses non-linear optimization to solve for the exact 3D camera orientation. This type of abstraction is very useful for authoring, since visual artists/cinematographers tend to think in terms of composing within the camera frame, not in terms of 3D camera parameters. We follow the methods of Drucker [13] and optimize over seven camera parameters using feasible sequential quadratic programming [14]. Although this method is susceptible to getting stuck in local minima, we have found that reasonable initial guesses allow the optimizer to converge quickly on good solutions. Also, since our approach to virtual cinematography is layered, shots with unsolvable constraints are avoided at higher layers as discussed above.

The system maintains low-resolution representations of the actors' bounding boxes that we call "blocking geometry," since only rough details of the geometry are required by the optimizer. This geometry consists of a set of hierarchically linked labeled 3D boxes that represent major anatomical landmarks on the character, such as the head, eyes, nose, chest, upper arms, hands, etc. Such geometry also exists for other objects in the environment.

In choosing the parameters of functions to optimize, the cinematographer agent considers the high-level tilt, orientation, and distance parameters specified by the director agent. For instance, if a low-angle medium close-up shot is specified, the agent might attempt to align the negative of the camera's look-at vector from fifteen to forty-five degrees below the direction that the front of the actor's body is pointing. In addition, parameters that specify positions in the frame are randomly "jittered" by a small amount of Gaussian noise so that shots do not appear rigidly identical. After optimization, the cinematographer agent will determine if the camera position is legal. A camera position is illegal if objects overlap the subject or a given part of it (e.g., a telephone pole occludes an actor's face) greater than a certain thresh-hold amount.

If the cinematographer agent discovers the camera position is illegal, it will then attempt to repair the shot. The agent uses a real-time algorithm to determine if objects are occluding the subject(s) of a shot and then attempts to remove offending objects. Basically, the idea is to only allow removal of objects in the scene if projections of the objects are contained entirely within the camera's frame and the objects cover a significant part of the frame. This is conceptually similar to methods commonly used in real cinematography and set design, in which scenes are "cheated" by rearranging pieces of the set, props, or actors to create the most aesthetically pleasing shot without revealing that the shot has been manipulated.

C. Editor agent

While the director plan/goal system is an efficient way to encapsulate high-level cinematic decisions, we use a different framework for dealing with global page layout issues. We also wanted to provide easy, global control for

the overall "mood" of the shots (e.g., attempt to favor low angles when an actor is frightened.) This led to our creation of an editor agent in the architecture that uses a simple biologically inspired "stimulation/inhibition" control framework. The basic concept of this framework is that various stimulator and inhibitor "hormones" interact with specific types of shots selectively but probabilistically. These hormones' strengths decay at different rates. Thus, the editor agent can globally affect which shots are picked by adding and removing hormones.

This control framework allows us to readily simulate some of the basic rules of comic book layout. One such rule is that adjacent panels (and to a somewhat lesser extent all panels on the same page) should not repeat shots with similar angles or cutting distance*s*. In order to do this, each shot adds some inhibitory hormones when it is used. For instance, over-the-shoulder shots add moderate inhibition for shots of the same distance and orientation (both have page long inhibition with decay, with the distance feature receiving stronger inhibition) and strong inhibition for over-the-shoulder type shots featuring the same speaker (which lasts only one panel.). In addition to the hormones added by shots, we also found the framework useful for other purposes. For instance, an inhibitor is always present to avoid shots with too much tilt. However, when an actor is frightened, stimulating hormones are added that can overcome this effect and favor titled off-axis shots.

D. Graphic Designer Agent

Each page holds a collection of *Panel* objects (i.e., comic book-style panels) and specifications for the panels are provided by the graphic designer agent. In the prototype implementation, when a new page begins, the graphic designer agent selects a page template from a database. This template then specifies the size and position of each panel. *Action!* can generate a reasonable image for almost any panel shape by using several approaches. Since all shots except special shots are dynamically rendered, the system can request that a given shot be rendered into any panel size. Additionally, certain types of shots can be inhibited for certain panel shapes (e.g., medium and long two-shots are inhibited when a panel has an aspect ratio less than 3:2.)

A final method for producing aesthetically pleasing panels of arbitrary shape is through cropping. Cropping is an important dramatic technique, which tends to focus the player on particular features of an image (e.g., a thin panel that just displays an actor's angry eyes.) The graphic designer agent can specify that a panel be cropped and crop centers for a given shot may be automatically generated by projecting certain features of the subject onto the camera's image plane, such as the center of the actor's head or eyes.

In addition to dynamically 3D rendered shots, there are also so-called special shots that make use of pre-rendered images. These shots use images created at high resolution and at several aspect ratios with pre-assigned crop centers. When a special shot is requested, the system retrieves the image from the database with the aspect ratio that most closely matches that of the panel and then resizes and crops the image as necessary. The concept behind special shots is that the author may want to achieve certain unique, predetermined effects at various points in the drama. These shots could include graphics such as special close-ups of objects, hand-painted panels, or comic book-like energy bursts.

IX. LITERARY RENDERING

The underlying concept behind literary rendering is that the dramatic narrative must have a point of view, which is determined both by external events as well as the internal state of the observer. Consider two characters, Marie and Carlos, and suppose that Carlos walks over to shake Marie's hand. If Marie has positive feelings toward Carlos, she may view his handshake as warm and friendly. On the other hand, if she detests him, she may equate his handshake to that of a used car salesman — insincere, greasy, and disgusting.

A. Dramatic writer agent

The dramatic writer agent contains *Descriptor* objects that are used to describe actions. At the point when a director agent chooses shots, it also decides what to describe. The director picks the appropriate *Descriptor* object based on the actor who is performing the action, the action, and the observer. The *Descriptor* object contains one or more *Features*, which describe a qualitative, observable way in which an action is performed (e.g., a hostile walk versus a seductive walk.)

The *Feature* object contains sets of string phrases, consisting of lists of verbs, adverbs, and adverbial phrases. The *Descriptor* object ranks all its *Feature* objects. In the prototype implementation, a feature determines its rank based on the intensity of a given actor expression (e.g., an actor's walk is predominantly angry.) The *Descriptor* then picks the a sentence template with the lowest score; each time a sentence is picked and rendered, its priority is incremented, which means that it will be less likely to be picked the next time.

As an example, in the prototype, we implemented a *Descriptor* object for describing Marie's observation of Carlos' moving around the set. The *Descriptor's Feature* objects differently describe Carlos' moving when his dominant expression is anger, passion, etc. For instance, positive verbs include "walking" and "striding", while negative verbs include "strutting" and "advancing." Negative phrases include "with his chest out, displaying the sort of arrogance that too many men confuse with

seduction" and "in a horrid attempt at appearing seductive"; positive phrases include "with the sort of unconscious masculinity that she cannot help but notice" and "with a grace that she does not expect from him." So, a rendered descriptive sentence may be "The soldier is striding into the verdant nightmare of the jungle, with the sort of unconscious masculinity that she cannot help but notice."

The dramatic writer agent also describes objects in the environment, the description varying according to the actor's emotional state. As an example, in the prototype the user can play a dentist, who will describe a dental chair as "the familiar, blue-green chair" or "the old dental chair" when positive emotions predominate. However, when negative emotions predominate the chair is described as "the broken catastrophe of a dental chair" or "the hideous blue green dental chair."

B. *Actor inner-voice*

As discussed, the actor emotion agent contains various *Emotion* objects, which in turn are linked to supporting *Episode* objects. Each *Episode* can be thought of as an actor inner-voice agent, capable of describing itself in a dramatic manner. Once the actor emotion agent decides to express an emotion, the highest priority *Episode* object is picked that has not already been used.

A sub-class of episode, the *ContrastEpisode* has the additional capability of returning a description that compares or contrasts itself to other episodes. As an example, we implemented a *ContrastEpisode* that describes Marie's attraction to Carlos. This episode is capable of *contrasting* or *synergizing* itself with other episodes. As an example, when one hatred episode is contrasted with a physical attraction episode, the following sentence can be produced: "Despite the heinous acts she imagines he could commit, she can think of nothing besides being in his arms." There is also an episode associated with gratitude, which may be formed if the soldier gives the dentist food. In this case, the attraction episode could synergize with the gratitude episode and the following sentence could be produced: "His beauty leaves her breathless, particularly after the small kindness of the banana."

X. EXECUTION PATH

Having described all of the component agents, we can now outline the execution path of the system:

1. The actor cognitive agent perceives (queries) the world.
2. Player goals, somatic goals, and emotional goals are executed (in that order.)
3. The actor emotional agent decays episodes.
4. Emotions to be expressed are selected (assuming emotions have attached episodes that have not been expressed.)
5. The director agent executes its plans/goals, which may result in sending a list of possible shots to the editor agent. The editor agent will then select possible shots, and send them to the cinematographer agent for lower-level processing. The cinematographer agent will take input from the graphic designer agent and ultimately send instructions to the 3D renderer so that images can be created. The director agent goals can also cause text to be generated by the dramatic writer agent.
6. The user interface is updated, based on the actor cognitive agent's altered perceptions.

XI. PROTOTYPE IMPLEMENTATION

A. *Back story*

We implemented a prototype drama designed for two players called *Dry Bones*. The drama is in the magic realist style, set sometime in the past in a tiny town lost in the jungles of South America. The first actor is Carlos, a guerilla fighting for the Conservative regime. For the last two weeks, Carlos has been plagued by a toothache. Lost from his patrol, Carlos is beginning to become delirious from the unbearable toothache. He stumbles upon the tiny village, and finds his way to the dentist's office. The second character is Marie, a young dentist volunteering with the Peace Corps. When the drama starts, she's been in the town for a year and witnessed a terrible massacre a few days ago, in which a whole family was wiped out, supposedly by the Conservative army. Shortly after this, she wakes up to find the entire town deserted. Stranded, she runs out of food within a few days. Just as she is sinking into complete despair, Carlos appears in the courtyard of her dental office.

B. *Platform details*

The prototype was implemented using Microsoft Visual C++ on the Windows NT platform. The client process is an ActiveX control embedded in an HTML page. The control can fire events that trigger Dynamic HTML code on the page, which allows us to implement user interface objects such as buttons and dialog boxes within the browser. The communications and render servers were implemented as stand-alone applications that communicate with the client processes using TCP/IP sockets. 3D graphics capabilities were implemented using the Win32 Open Inventor library (Template Graphics Software) and image processing was done with the LeadTools toolkit. When the render server receives a request, a JPEG image is rendered and the client can then retrieve it via a standard HTTP request.

C. Graphics

Courtyard and jungle environments were created as textured VRML objects. Some of the courtyard walls have arches and other openings — the system is aware that these do not cause occlusions, so it can create shots through the openings. High polygon models of the actors were used, and we created 17 variations of basic facial expressions (see Figure 3.) A wide variety of poses or basic stances that virtual actors may assume were also created (see Figure 4.) We also created high polygon models for various props such as the guerilla's gun, a dental chair, etc. These objects and the actors were used as sprites in the 3D world; sprites were pre-rendered at multiple angles with an alpha channel.

D. Prototype output

We present a few screen captures from prototype runs (see Figures 5-8.) The first screen shows the full interface. Icons in the lower frame are used by players to select actions (i.e., a chair and nitrous oxide tank are readily visible.) Overlapping this frame is a window that allows players to alter the actor's facial expression and to speak. The upper frame displays Marie's inner thoughts. Green text underneath the image panels is literary rendering created by the system and/or dialog from the players (some text is partially hidden by panels — players can see all the text by moving the mouse over it.) All of the captures except for the third are from Marie's point of view; captures were not retouched (except for the fact that the full interface is not shown in all), but were chosen to illustrate various capabilities and do not represent a continuous drama from a single session.

In the first capture, both actors are angry and the literary rendering reflects their hostile attitudes. The player controlling Carlos has chosen to hit Marie and the system has used a plan to depict this multi-part action. The first panel shows Carlos running toward Marie. The system has also chosen to crop this panel about a slightly offset center-of-interest (Carlos' face.) The second panel shows the two squaring off and the third shows the end result of the slap. Notice also how a dynamic, comic book-style "frozen action" effectively depicts the blow. In the next panel we see an enraged Marie and she is running off to get a gun. Notice how the system varies the cutting distance and angles to create variety.

In the second capture, both actors are very angry. To illustrate the actors' anger, the system has chosen a layout template with high, narrow panels. The system has also chosen close-up shots with cropping to heighten the sense of anxiety. In the final panel, Carlos draws his gun, which causes an episode to be triggered, Marie to become fearful, and the literary rendering at the top is generated. To emphasize the gun, the system chooses a close-up shot, which is a special pre-rendered image that is then cropped.

The third capture is from Carlos's point of view, and illustrates Marie about to drill on his teeth. He is in pain and frightened by the drill. Thus, the system is tending to favor tight shots to illustrate this tension. However, on the first panel, the director's goal to do a tight close-up on Marie's face was inhibited by its goal to show the menace of the drill. Note that we did not implement literary rendering for Carlos in the prototype.

In the final capture, Marie's positive emotions dominate since Carlos gave her food (previously in the drama) and the player has set Marie's expression to passionate. So, the literary rendering has an overall positive tone, and Marie's thoughts reflect her passion. The first four panels demonstrate the system's execution of a plan for filming a conversation that is interwoven with plans to establish the actors' spatial relation to each other.

XII. Conclusions and Future Work

Our prototype illustrates *Action!*'s ability to create interesting text and generate varied, aesthetically pleasing images that depict fairly complex sequences of actions. Moreover, it demonstrates a relatively easy to use and powerful interface. Many people who have played with the prototype have commented on the degree to which it seems "artistically" to choose shots and layouts. In fact, we have shown the system to a feature film director who claimed that many of the shots and sequences it generated are similar to ones he would have created.

However, there are several areas of the system that can be improved. For instance, a rendering system that does not use sprites and is capable of generating real-time animations could be used. One idea would be to animate the actors/camera within a panel for a few seconds, and then "freeze" the action and move onto the next panel. Instead of using pre-rendered expressions/poses, *Action!* could drive a lower level representation scheme, such as a real-time bones system. A moving camera would add greatly to the cinematic "feel" of the system. Support for non-rectangular panels and panels with transparency (alpha channels) that can overlap could also be added. This would give the system more flexibility in generating interesting layouts. Additionally, dynamic choices of panel sizes and positions would make the system even more interactive. In the prototype, a template is chosen for each page, which governs its design. Thus, layout/panel sizes on a single page are not varied to reflect dramatic shifts. Additionally, while we believe that the concept of literary rendering is very exciting, our current implementation is quite simplistic. Various natural language generation techniques can be applied to make this more flexible [15], [16].

We would like to acknowledge Jill Hunt for creating many of the beautiful graphics used in the prototype and Anne-Marie Gerber for all her love and assistance throughout this

project. This work is dedicated to Barry and Jane Gerber, who inadvertently started this whole thing by encouraging the reading of comic books on rainy days.

XIII. REFERENCES

[1] D. Arijon, *Grammar of the Film Language*, Communication Arts Books, Hastings House, New York, 1976.

[2] S. Katz, *Film directing shot by shot: visualizing from concept to screen*, Michael Weise Productions, Studio City, CA, 1991.

[3] R. Harvey, *The Art of the Comic Book: An Aesthetic History*, University Press of Mississippi, Jackson, MS, 1996.

[4] S. Lee and J. Buscema, *How To Draw Comics the Marvel Way*, Simon and Schuster, New York, 1978.

[5] J. Bates *et al*, *An Architecture for Action, Emotion, and Social Behavior*, Technical Report CMU-CS-92-144, Carnegie Mellon University, May 1992.

[6] K. Perlin and A. Goldberg, *Improv: A System for Scripting Interactive Actors in Virtual Worlds*, SIGGRAPH '96, pp.205-216.

[7] D. Rousseau and B. Hayes-Roth, *Improvisational Synthetic Actors with Flexible Personalities*, Stanford Knowledge Systems Laboratory Report KSL-97-10, 1997.

[8] L. He *et al*, *The Virtual Cinematographer: A Paradigm for Automatic Real-time Camera Control and Directing*, SIGGRAPH '96, pp.217-224.

[9] D. Kurlander *et al*, *Comic Chat*, SIGGRAPH '96, pp.225-236.

[10] R. Schank, *Dynamic memory: A theory of reminding and learning in computers and people*, Cambridge University Press, New York, 1982.

[11] A. Ortony *et al, The Cognitive Structure of Emotions*, Cambridge University Press, New York, 1988.

[12] W. Reilly, *Believable Social and Emotional Agents*, Ph.D. Thesis. Technical Report CMU-CS-96-138, Carnegie Mellon University, May 1996.

[13] S. Drucker, *Intelligent Camera Control for Graphical Environments*, Ph.D. thesis, MIT Media Lab, June 1994.

[14] C. Lawrence *et al*, User's Guide for CFSQP Version 2.5, Institute for Systems Research, University of Maryland, 1997.

[15] M. Kantrowitz and J. Bates, *Integrated Natural Language Generation Systems*, Technical Report CMU-CS-92-107, Carnegie Mellon University, April 1992.

[16] M. Walker *et al*, *Improvising Linguistic Style: Social and Affective Bases for Agent Personality*, Agents '97 Conference Proceedings, ACM, 1997.

Figure 3: Facial expressions.

Figure 4: Sample poses.

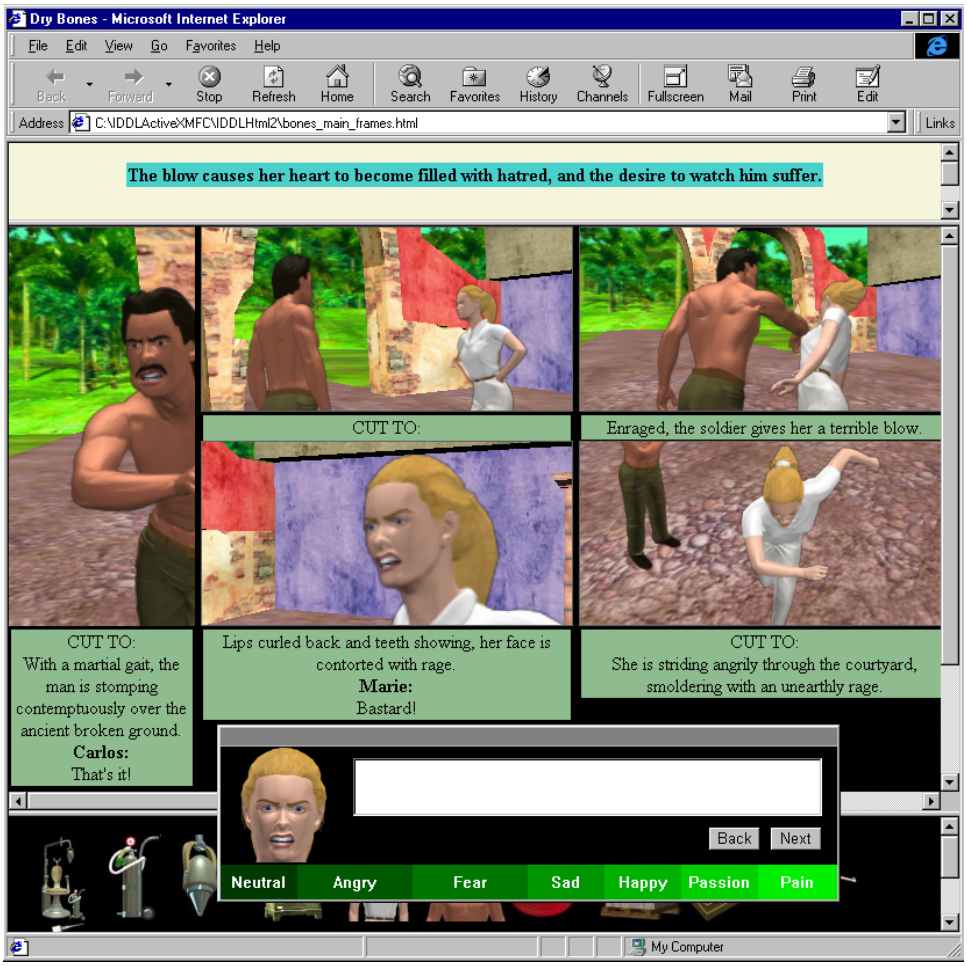

Figure 3: Prototype screen capture #1 (includes interface.)

Figure 4: Prototype screen capture #2.

Figure 5: Prototype screen-capture #3.

Figure 6: Prototype screen-capture #4.