Authoring Scenes for Adaptive, Interactive Performances

Patrick Gebhard Michael Kipp Martin Klesen Thomas Rist
DFKI GmbH
Stuhlsatzenhausweg 3
66123 Saarbrücken Germany
+49 681 302 5152
{gebhard, kipp, klesen, rist}@dfki.de

ABSTRACT
In this paper, we introduce a toolkit called SceneMaker for authoring scenes for adaptive, interactive performances. These performances are based on automatically generated and pre-scripted scenes which can be authored with the SceneMaker in a two-step approach: In step one, the scene flow is defined using cascaded finite state machines. In a second step, the content of each scene must be provided. This can be done either manually by using a simple scripting language, or by integrating scenes which are automatically generated at runtime based on a domain and dialogue model. Both scene types can be interweaved in our plan-based, distributed platform. The system provides a context memory with access functions that can be used by the author to make scenes user-adaptive. Using CrossTalk as the target application, we describe our models and languages, and illustrate the authoring process. CrossTalk is an interactive installation with animated presentation agents which "live" beyond the actual presentation and systematically step out of character within the presentation, both to enhance the illusion of life. The context memory enables the system to adapt to user feedback and generates data for later evaluation of user/system behavior. The SceneMaker toolkit should enable the non-expert to compose adaptive, interactive performances in a rapid prototyping approach.

Categories and Subject Descriptors
J.5 [**Arts and Humanities**]: Performing arts (virtual actors); H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems – *animations, evaluation/methodology*.

General Terms
Design, Experimentation.

Keywords
Authoring, user adaptivity, believability, embodied agents, virtual theater.

1. INTRODUCTION
Over the last couple of years, animated conversational characters have been used in a wide range of different application areas, including virtual training environments [22], interactive fiction [11][14] and storytelling systems [20], as well as in e-commerce applications where computer agents play the role of product presenters and sales assistants. Our work at DFKI builds on prior work on embodied conversational agents [7] and presentation agents [4]. One of the conversational characters developed at DFKI is Cyberella, a virtual receptionist which provides visitors with information about staff members and projects [8]. Cyberella assumes a setting in which the agent addresses the user directly like in human face-to-face conversations. However, there are situations in which direct agent-user communication is not necessarily the most effective and most convenient way to present information. Inspired by the evolution of TV commercials over the past 40 years, our group has discovered role-plays with synthetic characters as a promising format for presenting information. We have therefore proposed a shift from single character settings towards interactive performances by a team of characters as a new form of presentation [2]. The use of multiple characters allows to convey social aspects such as interpersonal relationships between emotional characters [16][19]. It also allows us to emulate small talk between characters which then becomes yet another performance or "meta-theater" [6]. The purpose for this "off-duty" activity, quite natural for humans, is twofold: (1) It attracts and binds the attention of passers-by, and (2) gives our agents the authenticity of real human actors, conveying the impression that they are permanently alive.

Using the theater as a metaphor we equated our agents with human actors. But where does the script come from which defines their verbal and nonverbal behavior? There are basically two approaches: The system can play the role of a playwright that automatically generates scenes at runtime [3] or we can use pre-scripted scenes that have been authored by a human writer. Ideally, an authoring system supports both scene types and provides creative experts (primarily non-programmers) with tools for the creation of rich, compelling content, and with an easy way to describe the *scene flow*, i.e. the transitions between scenes. The scene flow tells the system which scene should be played next during an interactive performance. The goal is to seamlessly integrate each scene into the overall script in order to obtain a believable result. This task is further complicated by the fact that in an interactive performance, the animated characters must be able to respond to the user in a way that is both appropriate and non-repetitive.

One of the first authoring systems for interactive applications was Improv [15] which consists of two subsystems. The first is an *animation engine* that uses procedural techniques to enable authors to create continuous motions and smooth transitions between them. The second subsystem is a *behavior engine* that allows authors to create sophisticated rules governing how actors communicate, change, and make decisions. The system uses an "English-style" scripting language to define individual scripts. A script is a sequence of commands that trigger specific actions or other scripts. In addition, Improv allows authors to create decision rules which determine the actor's tendencies toward certain choices over others. The overall behavior of an actor is determined by the script that is currently executed and by the non-deterministic behavior defined by the decision rules. As the number of scripts and rules increases it can therefore become more and more difficult to predict the runtime behavior of each character. Improv is a powerful authoring system but it does not provide support to deal with this complexity. Our system uses a similar scripting language to define individual scenes but we also provide an intuitive way to describe the scene flow. We believe that this makes it easier to produce a behavior which is consistent with the author's vision and intention. For character animation we currently use Microsoft Agent [13] with a fixed set of animations that cannot be modified at runtime.

Microsoft Agents are also used in SCREAM, a scripting tool which comprises modules for emotion generation, regulation, and expression [17]. In contrast to our system which uses an *author-centric* approach with the primary focus of scripting at the story/plot level, they use a character-centric approach in which the author defines an agent's initial goals, beliefs, and attitudes. These mental states determine the agent's behavioral responses to the annotated communicative acts he receives. SCREAM is intended as a plug-in to task specific agent systems such as interactive tutoring or entertainment systems for which it can decide on the kind of emotion expression and its intensity. The SCREAM system uses MPML, a Multimodal Presentation Markup Language which provides a visual authoring tool [21]. It was designed mainly for augmenting web pages with animated presentation agents similar to DFKI's WebPersona [4] and not for authoring interactive performances. As such, it does not have a context memory and only rudimentary support to define the scene flow, e.g. by using hyperlinks to jump to other parts in the script.

Neither Improv nor SCREAM allows interweaving of generated and pre-scripted scenes and despite their claim to provide an intuitive or English-style scripting language, both require a great deal of specialized programming expertise. We therefore propose a two-step approach: In step one, the scene flow is defined using cascaded finite state machines. In a second step, the content of each scene must be provided. This can be done either manually by using a simple scripting language, or by integrating scenes which are automatically generated at runtime based on a domain and dialogue model. Both scene types can be interweaved in our plan-based, distributed platform.

The rest of the paper is organized as follows. Section 2 describes the CrossTalk system which serves as our target application and major testbed for adaptive, interactive performances. Section 3 introduces the concept of cascaded finite state machines to show their use in authoring the scene flow and user interactions. Section 4 explains how to make an interactive performance user-adaptive and Section 5 concludes with a short description of our ongoing and future research.

2. THE CROSSTALK SYSTEM

CrossTalk is an interactive installation with animated presentation agents working with plan-based dialogue generation and a corpus of pre-scripted scenes (well over 220 scenes in German and English each). CrossTalk was first presented to the general public at the CeBIT convention 2002. The idea was to develop a new variant of information presentation in public spaces. As such, CrossTalk provides a spatially extended interaction experience by offering two separated agent screens, and by creating the illusion that the agents have cross-screen conversations. Hence the name "CrossTalk" [6][18].

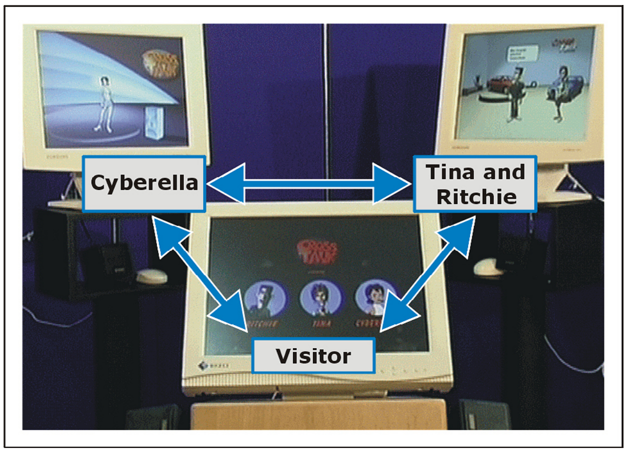

Figure 1: Main components and spatial layout of the CrossTalk installation.

Figure 1 shows the main components and the spatial layout of the CrossTalk installation. A visitor enters the CrossTalk installation by stepping in front of the user console (touch screen). The visitor is then welcomed by Cyberella (left screen) whose primary task is to play the role of a fair hostess and stage director for the two virtual actors Tina and Ritchie (right screen). Their role in CrossTalk is that of sales dialogue performers. In our scenario they play a salesperson and a customer that discuss the pros and cons of a car along a set of value dimensions (issues). The course and style of the conversation is influenced by a set of sales dialogue parameters which can be defined by the user prior to the performance:

- **Roles** – who is the salesperson and who is the customer?
- **Personalities** – polite vs. impolite, agreeable vs. disagreeable
- **Issues** – safety, comfort, prestige, operational costs, etc.

Cyberella first introduces the concept of simulated sales dialogues as a kind of personalized e-commercial, and then guides the user through a sequence of menus to specify the sales dialogue parameters for the next performance. She then uses these parameter settings to instruct Tina and Ritchie in her role as stage director (across screens) and starts the performance.

Now the visitor's attention is drawn to the "stage" (right screen) where the two actors Tina and Ritchie change their body postures to signal that they are now "on-duty". In the current scenario, a single car is discussed along the set of value dimensions (issues) specified by the user. Depending on their personality, the agents use different degrees of criticism (customer) and enthusiasm (salesperson) when talking about the car's features (consumption, horsepower, airbags, etc.). Clearly noticeable variations in the sales dialogues can be achieved because the personality settings determine both the dialogue strategies and the text templates used in the conversation.

During a performance the user can give feedback by pushing one of three buttons ("applause", "boo" and "help"). Such feedback may cause unexpected (meta-theatrical) behavior. For instance, if a visitor submits a "boo", the actors may get nervous and forget their lines. In contrast, "applause" makes them proudly smiling/bowing to the user. When "help" is requested, Cyberella stops the performance for short explanations.

After the performance, Cyberella takes over again, asking whether the user wants to see another sales dialogue, possibly with new settings. If not, the visitor leaves the installation and the actors go to "off-duty" mode adopting a more relaxed body posture. But instead of switching off or just idling around the agents display their off-duty behavior by chatting with each other across screens or by "rehearsing" for the next performance. The visitor is so encouraged to stay for awhile watching the "private lives" of the agents and, more important, new potential visitors are allured from the crowds of passers-by.

Since it's first presentation, CrossTalk has served its original purpose of attracting visitors at a number of occasions (CeBIT 2002, COSIGN 2002, IST Conference 2002, etc.). It is now our major testbed for adaptive, interactive performances.

3. AUTHORING INTERACTIVE SCENES

Authoring in CrossTalk is based on the concept of *scenes*. From the point of view of the system, scenes are pieces of user-edited contiguous dialogue (single utterances can also be scenes, not being a dialogue in the strict sense). From the point of view of the author, a scene is usually a coherent and closed unit regarding either a message, agent characterization or a humorous punchline.

Having assembled a huge corpus of pre-scripted scenes (more than 220 for English and German each) we realized that apart from the scripting [18] it is much more of a challenge to create, maintain and extend the *structure* of the story. Technically speaking, every story contains a logical scene flow that defines the transitions between scenes. This has to be modeled by the author and interpreted by the system at run-time. Usually, the scene flow is hard-wired and not reusable for other performances. Also, since stories are created by authors, possibly non-programmers, they depend on others to implement the story's logical framework.

In order to facilitate the story's implementation, we suggest a two-step approach for authoring interactive scenes. At first, the scene flow has to be defined using cascaded finite state machines. They refer to scenes whose content has to be provided in a second step. This content can be either pre-scripted scenes or scenes which are generated at runtime. A pre-scripted scene consists of pieces of dialogue – comparable to a screenplay – which include special tags to control the agent's non-verbal behavior, such as gaze, gesture, and body posture, as well as system control commands (see Figure 2). Scenes which are automatically generated at runtime, rely on a domain and dialogue model and may depend on user-defined parameters for the generation process. In a communication-theoretic view we consider the generation of simulated dialogues a plan-based activity [3]. First, we identify the basic dialogue moves in our domain (request, inform, etc.). Then, we define dialogue strategies to characterize typical combinations of dialogue moves. The strategies are encoded as plan operators that can be processed by the JAM agent architecture [10]. For each dialogue move a multimodal utterance is selected according to the context. The utterances contain text and gestures and were written by a human author using the same authoring syntax and gesture repertoire as for the pre-scripted CrossTalk scenes.

Sometimes we have to reuse the same scene over and over again. Cascaded finite state machines allow the shared use of modules (part of a scene flow), similar to subroutines in a programming language. This simplifies the modeling process in the case of, for instance, repeated patterns of agent-user interactions like simple yes/no questions. Also, such modules can be reused for other applications.

```
     Scene:  OFF-Chat stage-direction
        ...  ...
   Ritchie:  [TINA AS_LookLeft] Ok, if you are
             interested leave me your number.
             [V_LookToCy]
      Tina:  Well, <Pau=300> ok.
             [RITCHIE V_LookToActor]
             Sounds ... great. [AS_Glasses]
             I'll think about it.
 Cyberella:  [GS_Chide] My agent will contact you.
   Ritchie:  Yeah. Sure. [GS_DoubtShrug] All right.
```

Figure 2: Pre-scripted scene.

Our approach provides authors with a flexible toolkit for scripting scene flow, content and interaction. The SceneMaker toolkit complements the DialogueCompiler which transforms pre-scripted scenes to plan-based representations, as described in [6]. SceneMaker implements the techniques explained above for modeling performances.

3.1 Scene Flow as Cascaded FSM

Having written the content the author can define the narrative structure by linking the scenes in a graph called *scene flow*. Technically, we use cascaded finite state machines (FSMs) [9] to represent the scene flow. Cascaded FSMs are similar to Badler's parallel transition networks (PatNets) [5] used e.g. for the autonomous control of gaze and hand movement in animated agents. Compared to PatNets cascaded FSM do not allow the parallel execution of multiple actions (scenes). However, they allow simple scene flow management by providing hierarchical structures for a basically sequential process. Adding parallel structures would interfere with the simplicity of the authoring process.

A cascaded FSM consists of nodes and edges (transitions). Scenes can be attached to both nodes and edges. As mentioned above, scenes can be either pre-scripted or automatically generated. Both nodes and edges can have system commands attached, like accessing global variables or accessing context memory. This extends the scripting possibilities as it enables the creation of user-adaptive scenes (see Section 4).

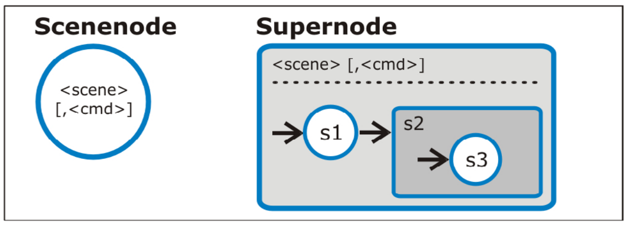

Figure 3: Node types.

A node represents a state in a performance. It can have a scene attached that is invoked by the system at runtime. There are two types of nodes (see Figure 3):

- **Scenenode**: Represents a state in which a pre-scripted scene is performed.
- **Supernode**: Represents a state in which a pre-scripted or an automatically generated scene is performed. Supernodes may contain sub-nodes of type supernode or scenenode. One of these sub-nodes must be declared the starting node. Edges connected to a supernode will be inherited by all sub-nodes.

At runtime, a node is called *running*, if the attached scene is currently performed, and *terminated* as soon as the attached scene is finished.

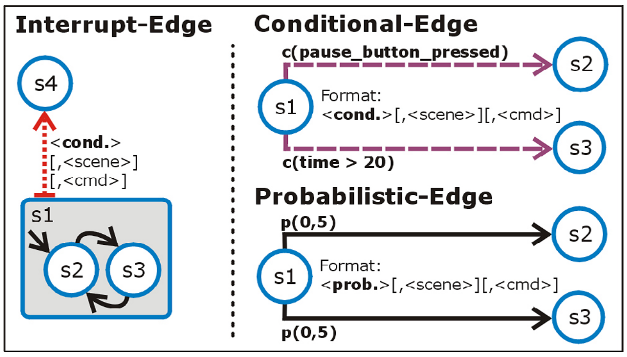

Figure 4: Edge types.

In our scene flow definition, edges define the transition between states. Like nodes they may have pre-scripted scenes attached which will be invoked when the edge is traversed. There are three types of edges (see Figure 4):

- **Interrupt Edge**: It is used for the handling of interruptive events, like pressing the pause button on a CD player. These edges have the task to directly interrupt a running node. Currently, this edge type supports temporal constraints like "20 seconds passed" or conditions like "user has pressed red button". If an edge points to its own source node, and if this node is a supernode, the author can specify to start over (i.e. restart interrupted scene) or jump to the last executed sub-node (i.e. resume interrupted scene).
- **Conditional Edge**: Only when the node is terminated, all conditional edges are checked in the order of author-specified priorities. This edge type supports the same constraints/conditions as the interrupt edge. If all conditions fail, the probabilistic edges are checked.
- **Probabilistic Edge**: These will be checked when the node is terminated and all conditional edges fail. In fact, probabilistic edges are ε-transitions tagged with a probability to support random branching in the scene flow.

Interrupt edges and conditional edges are mainly used for scripting the user interaction, whereas probabilistic edges are used as a design feature for making of the performance more variable.

A further enhancement for modeling the scene flow is a modular approach where supernodes can be used as subroutines. If an author wants to use this subroutine functionality (e.g. requesting user feedback) s/he has to define a *returning edge*. A returning edge is inherited by all sub-nodes of the supernode and can be of any of the above described edge types. By means of these edge types, you can determine how/when to jump back (e.g. user feedback received or time-out). Figure 5 shows the sharing of cascaded FSM supernodes as subroutines.

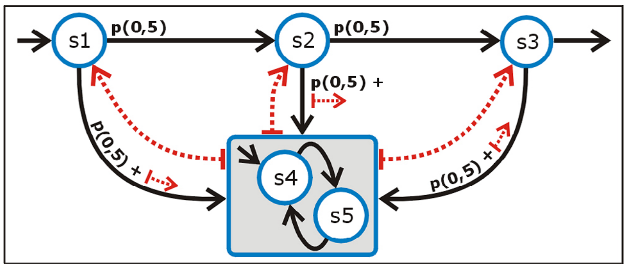

Figure 5: Supernodes as subroutines.

3.2 Scripting Scene Flow with Interaction

This section describes the scripting of the scene flow with user interaction using CrossTalk as an example. The major challenge is the modeling of asynchronous events as they occur in user interactions. Handling the user input can be modelled using various events and handling mechnisms:

- **Request and wait**: An agent asks the user a question and the system waits until the user answers.
- **Time-out events**: Waiting for user input, the system regains the initiative after a defined period of time.
- **Interrupts**: User feedback during the CarSales performance causes the system to seamlessly integrate a generic scene. Or, if the visitor leaves, the system interrupts all current activities and switches from on-duty to off-duty.
- **Concurrent event handling**: User feedback during the CarSales performance does not interrupt the performance but influences the agents' behaviour long-term by modifying the context.

These events and handling mechanism are implemented using the conditional and interrupt edge types. Request and wait is realized with conditional edges, one for each answer. Time-out events are produced using interrupt edges that act as observer demons and access system time. Both interrupts and concurrent event handling are realized in a similar manner. The latter but does not interrupt the current performance but modifies the context instead to achieve long-term changes in character behaviour.

With the use of the cascaded FSMs we were able to script the scene flow for the CrossTalk system. In the case of CrossTalk, we first specified the two major states on-duty mode and off-duty mode and the transition from one to the other. In a further step we refined the two modes in defining new sub-states and transitions between them.

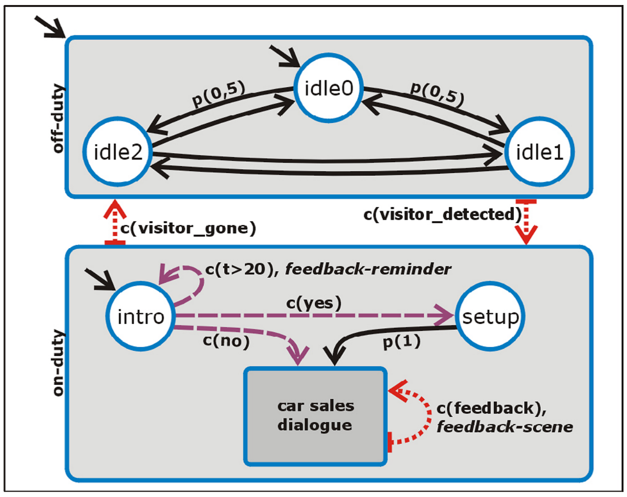

Figure 6: Simplified scene flow in the CrossTalk scenario.

Figure 6 shows a simple example in the CrossTalk scene flow: The launched system will start in *off-duty* mode modeled as a supernode with no scene attached (upper box). In a next step, the sub-node *idle0* that is declared starting node will be processed. After performing its attached scene, node *idle1* or node *idle2* will be processed with probability 0.5 each. If a visitor enters the CrossTalk installation, the currently processed sub-node of supernode *off-duty* will be interrupted using the interrupt edge *visitor_detected* and the scene *intro* in supernode *on-duty* will be performed. At the end of the scene, Cyberella asks whether the user wants to provide parameters for the ensuing demo. To handle this simple yes/no question (blocking event) we use a conditional yes-edge *c(yes)* and a no-edge *c(no)*. A third conditional edge *c(t>20), feedback-reminder* is triggered if the user does not answer within a certain amount of time (20 seconds). In this case the scene *feedback_reminder* is performed. During the presentation the user can give positive or negative feedback. This is realized using an interrupt edge *c(feedback), feedback_scene* which handles the feedback event. This event interrupts the generated sales dialogue and invokes the associated *feedback_scene*. Afterwards, the sales dialogue will be resumed. If the visitor leaves the CrossTalk installation, the interrupt edge *visitor_gone* immediately stops all ongoing activities in on-duty mode and activates the off-duty supernode.

3.3 SceneMaker Output

The SceneMaker generates a set of plans that can be interpreted by the JAM agent architecture [10]. The input consists of the scene flow, pre-scripted scenes, and automatically generated scenes, which must be in the JAM format. The scene flow is represented with the XML-based scene flow modeling language.

Figure 7 shows an excerpt of the scene flow described in the previous chapter. In the first step the pre-scripted scenes are translated by the DialogueCompiler into a set of scene plans, one for each scene [6]. In the following and final step, the SceneMaker creates the scene flow plans. The scene flow plans consist of scene plans, control plans and concurrent event processing plans. These plans, executed on our distributed, plan-based platform, run the show.

```
<scene-flow start="off-duty" name="simpleCrossTalk">
 <super-node name="off-duty" sub-start="node0">
  <i-edge target="on-duty">
   <conditions>
    <event name="visitor_detected" />
   </conditions>
  </i-edge>
  <scene-node name="node0">
   <scene name="idle0" />
    <p-edge target="node1" prob="0.5" />
    <p-edge target="node2" prob="0.5" />
   </scene-node>
   ...
 </super-node>
 <super-node name="on-duty" sub-start="node3">
  <i-edge target="off-duty">
   <conditions>
    <event name="visitor_gone" />
   </conditions>
  </i-edge>
  <scene-node name="node3">
   <scene name="intro" />
   <c-edge target="node4">
    <conditions>
     <event name="yes" />
    </conditions>
   </c-edge>
   <c-edge target="demo">
    <conditions>
     <event name="no" />
    </conditions>
   </c-edge>
   <c-edge target="node3">
    <scene name="feedback-reminder" />
    <conditions>
     <time-greater-than value="20" />
    </conditions>
   </c-edge>
  </scene-node>
  <scene-node name="node4">
   <scene name="setup" />
   <p-edge target="demo" prob="1" />
  </scene-node>
  <super-node name="demo">
   <automatic-scene name="car-sales-dialogue" />
   <i-edge target="demo">
    <scene name="feedback-scene" />
    <conditions>
     <event name="feedback" />
    </conditions>
   </i-edge>
  </super-node>
 </super-node>
</scene-flow>
```

Figure 7: Example scene flow specification.

4. MAKING CROSSTALK ADAPTIVE

Our aim to "attract and bind" the user is based on the two operational modes of the system: on-duty and off-duty. In off-duty mode the user sees three actors supposedly engaged in small talk and rehearsals, an unusual activity meant to "attract" a potential user. Having succeeded thus, the system enters on-duty mode and the user sees a performance as the result of the observed rehearsals. Now a systematic stepping out of character refers back to the agents' "real life" as actors that had been explicitly established by the off-duty mode. What we learned is that references to other parts of a story/scenario make a performance more complex and thus, more believable. Both should “bind” the user’s interest and keep him/her at the installation as long as possible.

To support the “binding” of the user we now aim at another point of reference besides the agents’ lives: the user him/herself. By tailoring the performers’ reactions to the user’s previous behavior we can convey the impression that our agents understand the user while at the same time making the user feel like having an impact on the presentation in a way that is subtle and long-term.

We go about this task by collecting information in a discourse history and filtering relevant data using measures like frequency and density. These measures are used to infer user stereotypes. Measures and stereotypes can be seen as a rudimentary user model. Both can be used by the author to make the selection of pre-scripted scenes context-dependent and user-adaptive.

On several occasions (see end of Section 2) hundreds of people have used the system, and interacted with it, often following certain patterns. It is obvious that this data should be fed back into the system to expand its possibilities and allow first, tentative evaluations. Therefore, we log discourse history data for further offline analysis.

4.1 Implementation

Our dialogue memory is modeled along the lines of the CrossTalk scenario. It is implemented as a object-oriented class hierarchy (Figure 8) where a session is the topmost concept. A session starts when a user arrives and ends when the user leaves. During one session several possible user interactions can occur:

- Yes/no question
- Personality choice
- Issues choice
- Role choice
- Yes question

A Yes question occurs when the user has been presented with a question or choice but did not answer within a pre-defined time span (time-out). Then, Cyberella will ask “Are you still there?” and a “Yes” button appears. This we call a Yes question.

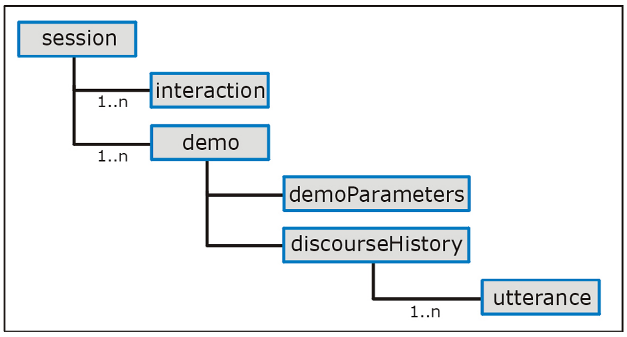

Figure 8: Context memory class hierarchy.

All interactions, with type and timestamp, are stored in the session object. It also contains an arbitrary number of demo objects which start when a user is shown a (automatically generated) demo and ends as soon as the demo ends (or the user leaves).

In CrossTalk, a demo consists of an automatically generated car sales dialogue. The possible user feedback (see Section 2) can be categorized into positive ("applause"), negative ("boo") and question ("help"). The context memory regards the agents’ contributions as well as the user feedback as *utterances*, storing speaker, addressee and content, e.g. if the user presses “boo” after an utterance by agent Tina, the context memory records (XML syntax):

```
<utterance speaker=”user” addressee=”Tina” content=”boo” />
```

For offline evaluation, the context memory writes log files to hard disk each time a session ends (see Figure 9).

```
<dialog-memory>
 <session start="1031125119212" end="1031125119212">
  <interaction-history>
   <interaction time="1031130327731" type="yesno"
                 responded="false"/>
   <interaction time="1031130526667" type="personality"
                 responded="false"/>
  </interaction-history>
  <demo start="1031125237752" end="1031125457178">
   <personality actor="ritchie" politeness="polite"
                 agreeableness="agreeable" />
   <personality actor="tina" politeness="polite"
                 agreeableness="agreeable" />
   <discourse-history>
    <utterance time="1031125260044"
                 speaker="visitor" addressee="tina"
                 strat="feedback" node="negative" />
    <utterance time="1031125303477"
                 speaker="visitor" addressee="ritchie"
                 strat="feedback" node="positive"/>
   </discourse-history>
  </demo>
 </session>
</dialog-memory>
```

Figure 9:Context memory sample log file.

4.2 Application

Of what use is the context memory to the author who strives to make the dialogues compelling and “personalized“? How can it be exploited for automatic dialogue generation and what kind of evaluation does it allow?

The authoring toolkit provides access functions to the context memory allowing to retrieve the following data:

- elapsed time (duration)
- feedback (positive, negative, question)
- interactions (type)

Feedback can be differentiated by addressee (Tina or Ritchie). For both feedback and interactions it is possible to request the number of occurrences and the number of time-outs. With this data, we are able to infer a user stereotype at runtime that can be used for conditional authoring. A first, tentative range of stereotypes includes: critical user (many negative feedback), active/passive user (many/few interactions), liking/disliking for agent X (many positive/negative feedback for agent X), lazy user (frequent interaction time-out), tenacious user (long sessions).

Besides the memory data and user stereotypes, we also suggest a range of measures that can be computed from the original data:

- feedback density (total/positive/negative)
- average response time (for each interaction type)

All the data can be used as conditions to trigger pre-scripted scenes or to control the generation process of generated dialogue. The access function values can also be inserted into pre-scripted scenes by means of placeholders (so agents can talk about the number of interactions etc.). To what effect this data can be used will be explained in the following sections.

4.2.1 Context in pre-scripted scenes

For the human author who tries to create generic dialogue pieces that nevertheless sound spontaneous and coherent, context knowledge allows him/her to react to unusual patterns, e.g. a positive response after a series of negative responses could trigger a side remark like "It was about time you said something nice!". Reacting to measures like feedback density could mean that if the density is too high, you need to ignore some feedback lest the presentation become too fragmented. Here are some other examples (X is either Tina or Ritchie, C is Cyberella):

- Liking for X → X says: "Oh, a real fan!"
- Disliking for X → X makes nervous gestures or does not react at all
- Disliking for X + positive feedback → X says: "That was about time!"
- Liking for X + negative feedback → X says: "Oops, just a little accident"
- Lazy user → C says: "You don't like talking, do you?"
- High feedback density → C says: "Why don't you just let it roll for a while" or start ignoring feedback.
- Liking could trigger some scene extensions, e.g. Tina/Ritchie giving out profuse thanks, side remarks by Cyberella about good vibrations, or by Ritchie on the topic of bribery...
- The user stereotype could guide the amount of options given to the user. If the user is active you could provide him/her with more frequent choices. If not, not.

The concept of Liking can trigger whole pre-scripted scenes (off-duty mode) where the agents argue about the worth of "popularity measured by empirical evidence" using data from context memory.

4.2.2 Context in generated scenes

If the generated content is semantically tagged (e.g., interesting vs. less interesting), one can guide the selection of plans (dialogue strategies) with the help of context. For instance, reacting to less interested users by playing more spectacular scenes or reacting to a long session duration by playing shorter scenes. The way you can use context in generated scenes is highly application-specific. If the agents are equipped with models of personality and emotions, the feedback can be processed by adapting the agents' mental state which in turn effects their behavior [1].

4.2.3 Context for Evaluation

Evaluation is a burning issue in the animated agents community [12]. Interactive systems have the great advantage that the interaction implicitly contains data about the system's effect on users. Measuring the time spent with the system alone allows tentative conclusions to be drawn about the attractiveness of the system. Using CrossTalk as a platform for arbitrary scenarios that are automatically generated, this kind of evaluation would allow to compare different character designs, different versions of a scenario, or whole dialogue generation engines – whether one is more interesting than the other.

5. CONCLUSION AND FUTURE WORK

In this paper, we introduced the SceneMaker toolkit that provides creative, non-programming experts with a simple scripting language for the creation of rich, compelling content. The scenes can be written using an ordinary word processor and are then translated by our DialogueCompiler into scene plans. This allows us to interweave such pre-scripted scenes with scenes automatically generated at runtime. We demonstrated that cascaded finite state machines are a powerful tool to define the scene flow in interactive performances because they allow authors to create transitions between scenes using probabilistic and conditional edges, states and sub-states, and to introduce concurrent events, e.g. to interrupt an ongoing scene and start a new one based on some user action. A further enhancement for modeling the scene flow is our modular approach where supernodes can be used as subroutines. This enables an author to reuse parts of the scene flow such as typical interaction patterns (e.g. yes/no questions, good-bye routine) in other applications. The SceneMaker also provides a context memory with access functions that can be used by the author to make scenes user-adaptive. It also functions as a logging facility for user profile generation and offline evaluation.

Currently, an author needs some programming expertise to define the scene flow with cascaded FSMs. In the near future we will provide a visual tool that allows direct manipulation of the nodes and edges representing the scenes and transitions. It will also allow to expand and collapse nodes when traversing the scene graph. On future occasions on fairs and conventions we will empirically test the use of context for authoring and evaluation. In the further development we will use the collected log files to create more stereotypes and extend our range of measures. Another aim is to create statistical models of behavior that can be used to predict future behavior. In CrossTalk, we can exploit this to predict interesting issues for the next demo that Cyberella can suggest ("Let me guess which issues would interest you most for the next demo"). Such statistical models can be differentiated using stereotypes.

We hope that our framework with its author-centric approach will contribute to the development of interactive applications in a variety of fields, including interactive cinema, virtual drama, e-commerce applications, without having to rely on low-level programming work.

6. ACKNOWLEDGMENTS

Our work builds on the contributions of our colleagues in the Intelligent User Interfaces department at DFKI: Stephan Baldes, Thomas Schleiff and Markus Schmitt. We would also like to thank our graphics designer Peter Rist for providing us with the virtual actors Cyberella, Tina, and Ritchie.

Our work is funded by the EU as part of the IST projects NECA, SAFIRA, MAGICSTER, and by the German Ministry for Education and Research as part of the MIAU and VirtualHuman projects.

7. REFERENCES

[1] André, E., Klesen, M., Gebhard, P., Allen, S., and Rist, T. Exploiting models of personality and emotions to control the behavior of animated interactive agents. In: Proc. of the Agents'00 Workshop on Achieving Human-Like Behavior in Interactive Animated Agents, 2000, 3-7.

[2] André, E., and Rist, T. Presenting through performing: On the use of multiple animated characters in knowledge-based presentation systems. In: Proc. of IUI'00, ACM Press, 1-8.

[3] André, E., Rist, T., van Mulken, S., Klesen, M., and Baldes, S. The automated design of believable dialogues for animated presentation teams. In: Cassell, J., Sullivan, J., Prevost, S., Churchill, E. (eds.). Embodied Conversational Agents, Cambridge, MA, The MIT Press, 2000, 220-255.

[4] André, E., Rist, T., and Müller, M. WebPersona: A life-like presentation agent for the world wide web. In: Proc. of the IJCAI'97 workshop on Animated Interface Agents: Making them Intelligent, Nagoya, 1997.

[5] Badler N., Webber B., Becket W., Geib C., Moore M., Pelachaud C., Reich B., Stone M. Planning and parallel transition networks: animation's new frontiers. In: Computer Graphics and Applications. Shin S. Y., Kunii T. L. eds. New York: World Scientific Publishing. 1995; 101-117.

[6] Baldes, S., Gebhard, P., Kipp, M., Klesen, M., Rist, P., Rist, T., and Schmitt, M. The interactive CrossTalk installation: Meta-theater with animated presentation agents. In: Proc. of the PRICAI'02 workshop on Lifelike Animated Agents, 2002.

[7] Cassell, J., Sullivan, J, Prevost, S., and Churchill, E. (eds.). Embodied conversational agents. The MIT Press, Cambridge MA, 2000.

[8] Gebhard, P. Enhancing embodied intelligent agents with affective user modelling. In: Proc. of UM'01 (doctoral consortium summary). Springer, Berlin, 2001.

[9] Harel, D. Statecharts: A visual formalism for complex systems. In: Science of Computer Programming, 8, 1987, 231-274.

[10] Huber, M. JAM: A BDI-theoretic mobile agent architecture. In: Proc. of the Third Conference on Autonomous Agents, ACM Press, 2001, 236-243.

[11] Laurel, B. Computers as theatre. Addison-Wesley, Reading MA, 1993.

[12] McBreen, H.M., Anderson, J.A. and Jack, M.A. Evaluating 3D Embodied Conversational Agents In Contrasting VRML Retail Applications. In: Proc. of the Agents'01 Workshop on Multimodal Communication and Context in Embodied Agents, 2001, 83-87.

[13] Microsoft Agent Software Development Kit, Microsoft Press, Redmond, WA, 1999.

[14] Murray, J.H. Hamlet on the holodeck: The future of narrative in cyberspace. The MIT Press, Cambridge MA, 2000.

[15] Perlin, K., and Goldberg, A. Improv: A system for scripting interactive actors in virtual worlds. Computer Graphics, 29 (3), 1996.

[16] Prendinger, H. and Ishizuka, M. Social role awareness in animated agents. In: Proc. of the Fifth Conference on Autonomous Agents, ACM Press, 2001, 270–377.

[17] Prendinger, H. and Ishizuka, M. SCREAM: Scripting emotion-based agent minds. In: Proc. of AAMAS'02, ACM Press, 2002, 350-351.

[18] Rist, T., Baldes, S., Gebhard, P., Kipp, M., Klesen, M., Rist, P., and Schmitt, M. CrossTalk: An interactive installation with animated presentation agents. In: Proc. of COSIGN'02, 2002.

[19] Rist, T., Schmitt, M. Avatar arena: An attempt to apply socio-physiological concepts of cognitive consistency in avatar-avatar negotiation scenarios. In: Proc. of AISB'02 Symposium on Animated Expressive Characters for Social Interactions, London, 2002, 79-84.

[20] Ryokai, K., Vaucelle, C., Cassell, J. Virtual peers as partners in storytelling and literacy learning. In: Journal of Computer Assisted Learning (in press).

[21] Saeyor, S., Binda, H., and Ishizuka, M. Visual authoring tool for presentation agent based on multimodal markup language. In: Proc. of Fifth International Conference on Information Visualization (IV'01), IEEE, 2001, 563-570.

[22] Traum, D., and Rickel J. Embodied agents for multi-party dialogue in immersive virtual worlds. In: Proc. of AAMAS'02, ACM Press, 2002, 766-773.