M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 391
To appear in CHI'97 — October 1996

Interval Scripts: a Design Paradigm for Story-Based Interactive Systems

Claudio S. Pinhanez *
MIT Media Laboratory
pinhanez@media.mit.edu

Kenji Mase
ATR–MIC Research Lab.
mase@mic.atr.co.jp

Aaron Bobick
MIT Media Laboratory
bobick@media.mit.edu

Abstract

A system to manage human interaction in immersive environments was designed and implemented. The interaction is defined by an *interval script* which describes the relationships between the time intervals which command actuators or gather information from sensors. With this formalism, reactive, linear, and tree-like interaction can be equally described, as well as less regular story and interaction patterns. Control of actuators and sensors is accomplished using PNF-restriction, a calculus which propagates the sensed information through the interval script, determining which intervals are or should be happening at each moment. The prototype was used in an immersive, story-based interactive environment called *SingSong*, where a user or a performer tries to conduct four computer character singers, in spite of the hostility of one of them.

1 Introduction

The objective of this research is the design and implementation of an interaction manager system which is able to handle complex patterns of interaction evolving through time. The interaction manager should be able to track multiple concurrent stories, turned on or off according to the development of the story and the users' actions. For example, the user or users can be interacting with a virtual characters of the story, while the characters are also engaged in interaction among themselves.

We are particularly interested in developing a paradigm for scripting story-based interactive systems which can handle progression through time. That is, we want the behavior of the characters and the development of the story to depend on the past interaction. This paper describes a script paradigm based on Allen's time intervals ([1]) which can be employed by an interaction manager by using the PNF calculus of Pinhanez and Bobick ([6]).

We start by detailing some of the fundamental theoretical concepts behind the proposed script paradigm and its realization in the interaction manager. We then describe some of the issues involved in the implementation of such interaction manager, and our experience using the manager in a real system.

*This research was conducted at ATR Media Integration & Communication Research Laboratories; the author was supported by a Starr grant from the MIT/Japan Program nand by ATR Research Laboratories.

2 Motivations

A long term objective of this research is to create immersive, interactive environments which capture the intensity and dramaticity of good stories. These environments can be either experienced directly by the user, as in the *interactive cinema* concept proposed by Tosa and Nakatsu ([8]); or employed in a computer theater performance, as described by Pinhanez [5]. With only a few exceptions (for example, [3]), immersive interactive environments have been *exploratory*, i.e., the user's basic objective is to navigate through an artificial world, discovering its interesting features and/or meeting with virtual creatures. We believe that immersive systems can be significantly enriched by incorporating dramatic structure from stories.

To enable the design of such environments, many technological developments are necessary: improvement of sensing technology, human action understanding, wireless interfaces, etc. But it is also fundamental to develop paradigms and tools for scripting systems and stories able to handle a variety of interaction situations. Most story-based environments till now have relied on scripts which are either *reactive* or shaped in a *tree-like* structure. In *reactive* systems, the story (if it exists at all) unfolds as a result of the firing of behaviors as a response to the user's actions (for example, [4, 7]). In *tree-like* scripts, the user typically chooses between different paths in the story through some selective action (for example, [3]).

Those scripting methods are not good for describing and managing the complex interactivity we are planning for our future immersive environments. It is hard to express progression of time in creatures controlled by reactive systems, and handling parallel events in a tree-like, multiple choice script is cumbersome. In the following section, we propose a scripting paradigm which has the potential to handle multi-pattern interaction in story and scenario-based interactive environments.

3 Interval Scripts

An *interval script* is a low level interactive script paradigm based on explicit declaration of the relationships between the time intervals corresponding to actions and to sensor activities. In an interval script, the designer of the interactive system declares the time intervals corresponding to the different actions and events and the time relationship between some of those pairs of intervals, i.e., whether two intervals happen in sequence, overlap, or are mutually exclusive. No explicit time references are needed, for either duration, start, or finish of an interval. Examples of interval scripts are provided later in this paper.

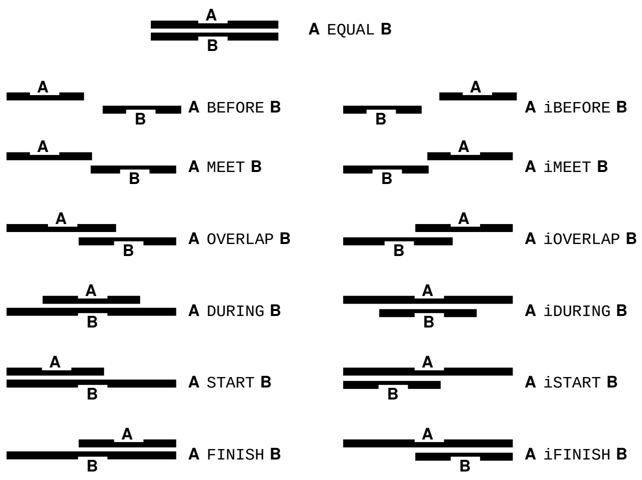

Figure 1: The possible 13 primitive time relationships between 2 intervals [1].

3.1 Allen's Interval Algebra

To model the time relationships between two intervals we employ the interval algebra proposed by Allen [1]. The interval algebra is based on the 13 possible primitive relationships between two intervals which are summarized in fig. 1. In any actual situation, two intervals relate to each other exactly as described by one of the possible primitive time relationships.

Given two situations in the real world, their possible time relationship can always be described by a disjunction of the primitive time relationships. For instance, we can say that the action of driving a car either STARTS or FINISHES or happens DURING or is EQUAL to the interval when the car engine is turned on. That is, the time relationship between driving and having a car engine on can be described by the disjunction of {`START`, `DURING`, `FINISH`, `EQUAL`}. Of course, in a real occurrence of a driving action, only one of the relationships actually happens.

Most of the interest in Allen's representation for time intervals comes from a mechanism by which the time relationships between the pairs of intervals can be propagated through the collection of all intervals. For instance, if interval `A` is `BEFORE` `B`, and `B` is `BEFORE` `C`, Allen's representation enables the inference that `A` is `BEFORE` `C`. In fact, [1] provides an algorithm, later revised by [9], which propagates the time relations through a collection of intervals, determining the most constrained disjunction of relationships for each pair of intervals which satisfies the given relationships and is consistent in time.

There are many reasons to use Allen's algebra to describe relationships between intervals. First, no explicit mention of the interval duration or specification of relations between the interval's extremities is required.

Second, the existence of a time constraint propagation algorithm allows the designer to declare only the relevant relations, leading to a cleaner script. Allen's algorithm is able to process the definitions and to generate a constrained version which defines only the scripts which satisfy those relations and are consistent in time.

Third, the notion of disjunction of interval relationships can be used to declare multiple paths and interactions in an story. As we mentioned before, any instance of an actual interaction determines exactly one relationship for each pair of intervals. Thus, we can see the interval script as the declaration of a graph structure where each node is an interval, and whose links are constrained by the structure of time. An interval script describes a space of stories and interactions.

Fourth, it is possible to determine whether an interval is or should be happening by properly propagating occurrence information from one interval to the others as described in the next section. In other words, it is possible to construct an interaction manager which takes relationships between intervals as a description of the interaction to occur and which by getting input from sensing routines can determine which parts of the script are occurring, which are past, and which are going to happen in the future.

4 Run-Time Management of Interval Scripts

Allen's interval algebra describes a way by which the relationships between intervals can be propagated by a transitive rule. To use the relations between intervals in a system to manage real-time interaction we employ the *PNF calculus* developed by Pinhanez and Bobick [6] which defines a method for propagating occurrence information through a network of intervals.

Although the `PNF` calculus makes the essential link between the concept of interval and current time, all the `PNF`-related concepts are completely transparent for the designer of an interactive application: an interval script talks about the activity of actuators and sensors in the real world, and how they interact locally. However, to understand how it is possible to represent and detect the current situation within a script given the input of some sensors and the information about the past interaction, we describe briefly the principal concepts of the `PNF` calculus in the following paragraphs.

4.1 `PNF` Calculus

The `PNF` calculus is based on the assignment of a primitive state value, either past (`PAST`), now (`NOW`), or future (`FUT`), to each interval at each instant of time. Those values correspond to the intuitive notion of interval occurrence relatively to a given moment of time. To match the structure of the interval relationships (which uses disjunctions of primitive relationships), the current state of each interval can be characterized by a disjunction of possible primitive states, a `PNF`-*state*. There are 7 possible `PNF`-states: `PAST`, `NOW`, `FUT`, `PN`, `PF`, `NF`, or `PNF`.

For instance, `PN` (`PAST` or `NOW`) describes the situation where it is known that the interval started some time in the past but it is unknown if the interval has already finished. `PNF` (`PAST` or `NOW` or `FUT`) stands for the situation where no information about the interval is presently known.

The `PNF`-restriction algorithm developed by Pinhanez and Bobick [6] enables the propagation of known `PNF`-states of some intervals through a network of intervals. Typically in the cases described in this paper the current `PNF`-state

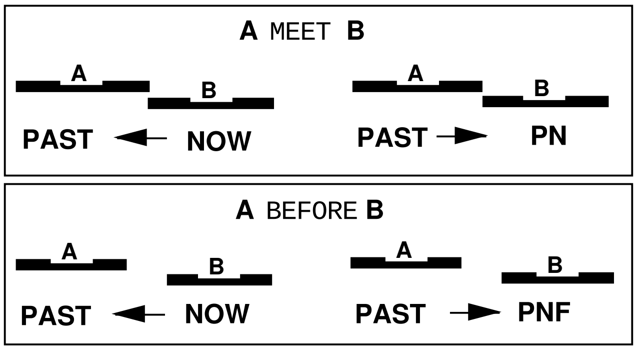

Figure 2: Examples of PNF value propagation using the PNF-restriction algorithm. Two different cases are exemplified: in the first, A `MEET` B; in the second, A `BEFORE` B.

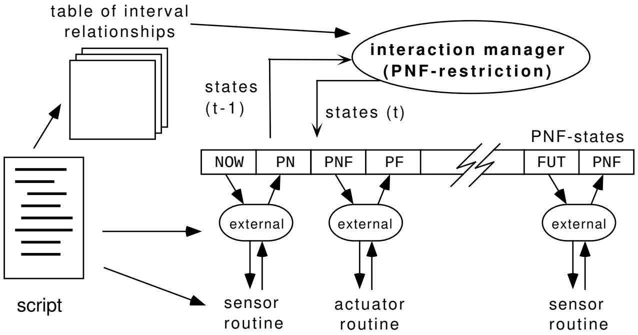

Figure 3: Diagram of the interaction manager.

of some intervals are obtained by sensor devices. By using the `PNF`-restriction algorithm, the `PNF`-state of intervals related to actuators can be determined. Actuators in the `NOW` state are activated, and those which are in the `PAST` state are disabled.

Figure 2 shows some simple examples of propagation of values considering the relationship between two intervals A and B. In the first case, A `MEET` B: if B is `NOW` then it is clear that A is in the past (considering that intervals are supposed not to contain the endpoint). However, if it is known that A is `PAST`, then we can only conclude that B is `PAST` or `NOW` that is, `PN`. In the second case, if A `BEFORE` B, the information that A is `PAST` is virtually useless, since B may have already happened (`PAST`) or be happening (`NOW`) or be in the future (`FUT`) characterizing a `PNF` state.

In [6], Pinhanez and Bobick also discuss the idea of *time expanding* a `PNF`-state. Basically, given the `PNF`-state of an interval, the time expansion of the interval is the `PNF`-state corresponding to possible states of that interval in the next instant of time. If an interval is `NOW`, in the next instant it may be `NOW` or `PAST`, or `PN`; if it is `PAST`, it remains in `PAST`; if it is `FUT` it goes to `NF`. A comprehensive description of the `PNF`-restriction algorithm can be found in [6], together with some theorems about its computational complexity and completeness.

4.2 Connecting to the Real World

According to our proposal of interval scripts, the interaction of a system is described by intervals of time and their relationships. Some intervals are connected to sensors and some are connected to actuators. Connectors with real world events are generally referred as *externals*. In the interval script paradigm, the designer has two basic tasks: to define the actual sensing and actuating routines corresponding to different externals and to determine the relationships between the intervals defined by those externals.

An *external* is the concept abstracting the internal mechanisms required to run the different sensors and actuators. In fact, an external seamlessly encapsulates the connections between a designer's routine and the `PNF` structure used to manage the interaction. Quite commonly more than one interval is associated to one external, as shown later.

Figure 3 shows the basic structure of the interaction manager. The script defines the relationships between intervals, which are stored in a table, and used by the interaction manager when running the `PNF` algorithm. The interaction manager considers the `PNF`-state of all intervals at time $t-1$ to compute the `PNF`-states at time t. These values are converted — as discussed later — and used to call the designer's sensing and actuating routines. The outputs of those routines are mapped back into `PNF`-states of appropriate intervals, completing the cycle.

Sensors

In interactive environments, sensors can play the roles of chooser, locator, valuator, etc. (see [2]). We have analyzed and implemented only the binary case of a chooser sensor, that is, a sensor which detects whether something is happening or not. However, all sensors have at least two time intervals naturally associated to them: an *activity* interval which determines when the sensor is active, and a *event* interval which corresponds to an occurrence of the sensor.

In the case of binary-choosing sensors, the designer of the interactive system has to provide a routine which receives as input a switching command (`ON`, `OFF`, `RESET`) and returns one of the following 3 values: `HAPPENING`, `NOT-HAPPENING`, or `UNKNOWN`. During the time the activation interval is or may be happening (`NOW`, `PN`, `NF`), the interaction manager sends an `ON` command to the designer's routine, and `OFF` otherwise.

The output of the sensing routine affects the state of the `event` interval as follows:

`HAPPENING`: the `event` interval is set to the `NOW` state by the interaction manager;

`NOT-HAPPENING`: the `event` interval is set to `PF`;

`UNKNOWN`: the `event` interval is set to `PNF`.

Figure 4 shows the intervals associated with a sensor. In the top example, it is shown that the `event` interval can occur many times while the `activity` interval is in the `NOW` state (and therefore, sending `ON` messages to the sensing routine). The second example shown in the bottom of fig. 4 exemplify the case of *triggers*, which turn on only once; this is achieved by automatically incorporating into the script the relationship stating that the `activity` interval `MEET` the `event` interval.

Actuators

In the case of actuators, the designer has to provide a routine which accepts a switching command (`ON`,

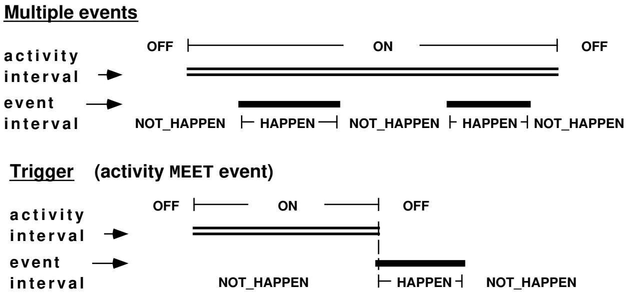

Figure 4: Intervals associated with a sensor, in two different configurations.

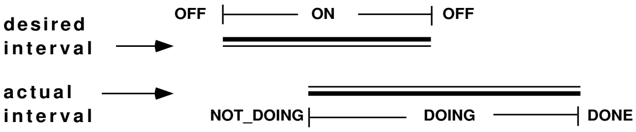

Figure 5: Intervals associated with an actuator.

OFF, RESET) and returns a state-descriptive message: NOT-DOING, DOING, or DONE. The feedback from the routine is important because actions in the real world have their own timing and priority, independent of the desires of the designer or of the script. A situation might call for the playing of a sound, but the sound might be delayed by a network problem or might not happen at all, if, for instance, another actuator has already grabbed some required hardware connection.

These characteristics of actuators are reflected in our system by associating two intervals to actuators: a *desired* interval and an *actual* interval. Figure 5 shows the relationship between the intervals and the actuator routine provided by the designer. When the desired interval is happening (i.e., has value NOW), the interaction manager sends ON messages to the designer's routine. When, and if the actuator goes on, the returning DOING message moves the actual interval into the state NOW.

When the desired interval goes to PAST, the interaction manager starts sending an OFF message to the actuator routine. However, the end of the actual interval is decided by the routine itself: it may happen before, at the same, or some time after the routine receives the OFF message, depending on the properties of the device being controlled.

Another issue is what to do when the PNF-state of the desired interval is PN, NF, PF, or PNF: should an ON or an OFF message be sent to the designer's routine? Our current solution is to make the designer define the desired interval to be RELAXED or ANXIOUS. In the former case, only pure NOW PNF-states send an ON message; in the later, any state containing NOW — except PNF— starts the actuator's routine.

Timers

Although the general objective of this proposal is to write a script without explicit time references, sometimes it is necessary to constrain the duration of an action or a sensing activity. In our conceptualization, a timer is a special case of an actuator, thus defining desired and actual intervals. The desired interval is used to turn the timer on and off; the actual interval — especially its end — can be used to trigger another actions as the timer expires.

4.3 Running Cycle

Before the interaction manager can actually run the interaction described by the interval script, Allen's algorithm must be executed to assure that the relationships between every two intervals is as restricted as possible.

Just before the interaction starts, the interaction manager sets every interval state to PNF, except for the special interval *start*, which is assigned the value NOW. During run-time, the following basic cycle is repeated till the special *end* interval becomes NOW:

1. at the beginning of each cycle, all designer's routines connected to externals are called, considering the PNF-state of each interval to decide which of the switching commands (ON, OFF, RESET) is passed to the routine.
2. the outputs of the designer's routines, translated into PNF-states are attributed to the appropriate interval connected to each external;
3. intervals which are not connected to externals (if they exist) have the values in the previous iteration *time-expanded.*
4. the PNF-restriction algorithm is applied, propagating the current values of some intervals through the whole network;
5. if the *end* interval is not NOW, the cycle is repeated.

According to this cycle, if an interval becomes PAST, it remains with this value until the end of the run. This information is used in the upcoming cycles, constraining the values of other intervals and making the system progress through the story defined by the interval script.

4.4 Implementation

Allen's interval algebra, the PNF-restriction algorithm, and the "external" procedures have been implemented in C++. In the current version of the interaction manager, the designer's routines are also written in C++, including the declaration of externals and their relationships. The interval script is simply a C++ file which uses classes corresponding to the different externals.

Since the computational complexity of the PNF-restriction algorithm is worst-case quadratic in the number of intervals — and linear in memory usage — ([6]), the interaction manager can run the PNF-restriction algorithm at intervals compatible with sensor accuracy. Typically, we have been running a new cycle of the manager at video rates, that is, 30 times a second.

5 An Experiment: SingSong

The methodology and algorithms described in this paper were tested in a story-based, interactive system named *SingSong*. *SingSong* was designed to be enjoyed both as an user experience and as a computer theater performance. In the later case, as described by Pinhanez in [5], our system provides computer-generated partners to the human performer which are not only reactive, but are also able to follow the script.

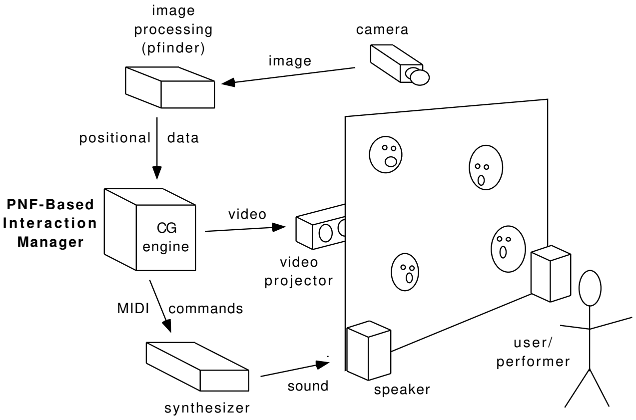

Figure 6: The basic setup of *SingSong*.

The transition between the performance and the user mode is seamless, enabling the user to experience the story as lived by the performer. Typically, a performer is able to produce a more vivid and interesting result for those observing from outside the system because she can clearly react to the situations, and expressively displays her emotions.

Figure 6 shows the basic structure of *SingSong*: a big video screen shows four computer graphics-animated characters which can "sing" musical notes (as produced by the synthesizer). A camera watches the user or performer, determining the position of her head, hands, and feet. Those positions are recovered by the software *pfinder* developed at the MIT Media Laboratory ([10]).

All the interaction is nonverbal: the user or performer gestures and the CG-characters sing notes and move. There is a CG-object — a pitching fork — which the user employs during one of the scenes. Sounds of applauses can also be generated by the system. *SingSong* is an environment which immerses the user or performer in a simple story which unfolds as the interaction proceeds:

> *Singers of a chorus (the CG-creatures) are animatedly talking to each other. The conductor enters, and commands them to stop by raising her arms. One of the singers — #1 — keeps talking, till the conductor asks it to stop again. Singer#1 stops, but complains (by expanding and grudging sounds). The pitching fork appears and the conductor starts to tune the chorus: she points to a singer, and "hits" the pitching fork by moving her arm down. Any singer can be tuned at any time. However, singer#1 does not get tuned: it keeps giving back the conductor a wrong note till the conductor knees down and pleads for its cooperation. After all the singers are tuned, a song is performed. The conductor controls only the tempo: the notes are played as she moves her arms up. When the song is finished, applause is heard, and when the conductor bows back, the singers bow together with her. Just after that, singer#1 provokes the conductor again, and the singers go back to talking to each other.*

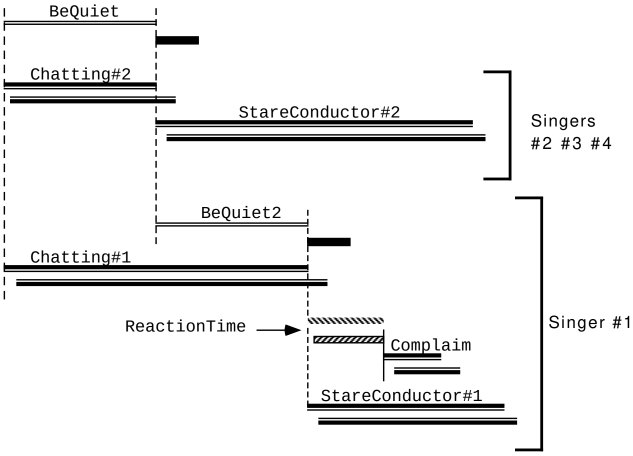

Figure 7: Diagram of the temporal relations in the first scene of *SingSong*.

5.1 Writing An Interval Script for SingSong

SingSong portrays a very simple, quasi-linear story. The choice of a story with some linearity was intentional: interval scripts are naturally fit to describe reactive environments, and we wanted to test the limits of the concept in a very constrained interactive story. The following paragraphs show some examples of how semi-linear and parallel structures can be described by the interval script approach.

Commanding Multiple Characters

In the first scene of *SingSong*, singer#1 has a different role than the other singers: it does not obey the conductor promptly, and after being commanded to stop, it complains. After the singers stop chatting, they start to stare to the conductor following him around the space. As we see, there are several externals involved in this scene:

`Chatting` (actuator): 4 copies, one for each singer, controls the sound and the mouth movements which simulate chatting;

`BeQuiet` (trigger sensor): fires if the user raises both arms on top of his head;

`BeQuiet2` (trigger sensor): identical to `BeQuiet`;

`StareConductor` (actuator): 4 copies, makes the eyes of the creatures follow the conductor around the space;

`ReactionTime` (timer): provides a pause of 3 seconds between the conductor's gesture and the complaint;

`Complain` (actuator): only for singer #1, controls the sounds and graphics related to the complaining action.

Figure 7 displays a diagram showing how the different intervals of each external are related. The diagram shows the relationships for singer#1 and singer#2; the relationships for the other two singers are identical to those of #2. Figure 8 shows the interval script corresponding to the first scene (where the details of the `C++` code were omitted for clarity).

Initially, the `desired` interval of all `Chatting` actuators and the `activity` interval of `BeQuiet` are defined to

BeQuiet activity	START EQUAL iSTART	Chatting#1 desired
BeQuiet activity	START EQUAL iSTART	Chatting#2 desired
BeQuiet event	MEET BEFORE	Chatting#2 desired
BeQuiet event	MEET BEFORE	BeQuiet activity
BeQuiet event	START EQUAL iSTART	StareConductor#2 desired
BeQuiet event	START EQUAL iSTART	BeQuiet2 activity
BeQuiet2 event	START EQUAL iSTART	ReactionTime desired
BeQuiet2 event	START EQUAL iSTART	StareConductor#1 desired
Reaction Time event	MEET	Complaim desired

Figure 8: Interval script corresponding to the first scene of *SingSong*.

start together. This is shown in the first part of fig. 8 which states that the `activity` interval of the sensor `BeQuiet` `START` or `EQUAL` or `iSTART` the `desired` interval of `Chatting#1` and `Chatting#2`. In fig. 7 we represent this definition by the dashed line joining the beginning of both intervals. The beginning of these intervals is triggered by other intervals, from the prologue, not shown here.

The next definition states that `BeQuiet.event` finishes the `desired` interval of `Chatting#2`, i.e., the singers stop chatting when the conductor raises his arms. The `event` of `BeQuiet` also turns off the `activity` of `BeQuiet`, what makes this external a trigger sensor. Also, this event starts `StareConductor#2.desired`. The turning on and off of intervals is described by the `START` or `EQUAL` or `iSTART`, and the `MEET` or `BEFORE` relationships, respectively, as shown in fig. 8.

However, since singer#1 does not stop chatting till the conductor raises his arms for a second time, `BeQuiet.event` does not neither turn off `Chatting#1` nor turn on `StareConductor#1`. Instead, `BeQuiet.event START` or `EQUAL` or `iSTART` — i.e., triggers — `BeQuiet2.activity`.

A detection of an `event` by `BeQuiet2` shuts off the sensor's activity, starts the `StareConductor#1`, and the `desired` interval of timer `ReactionTime`. The end of `ReactionTime.actual` starts the `Complain` actuator, finishing the first scene.

As it can be seen in fig. 8, all the structure is described by the time relationships between intervals, and there are no explicit references to duration of intervals. This scene of *SingSong* is a good example of parallel actions that start from a single event.

Other Possibilities

The complete script of *SingSong* includes many different constructions which are handled conveniently by the time interval relationship paradigm. The detailed examination of those constructions are beyond the scope of this paper. However, it is useful to mention typical cases of interaction which were addressed during the development of *SingSong*.

The tuning scene is basically a loop of short tuning interactions between the conductor and one of the singers, until all the singers are tuned. This is handled by a third interval associated with all externals, the *reset* interval. Whenever a `reset` interval happens (i.e., the interval has a `NOW` `PNF`-state), the other intervals associated with that particular external are set to `PNF`. Basically, this enables a movement backwards in time which, when coupled with a sensor of loop termination, makes the construction of loops possible.

Another interesting case occurs in the singing scene. The singing of the melody is implemented as a loop where each note is triggered by an upwards movement of the conductor's arms. However, to avoid breaking the flow of the melody, a timer starts in parallel with the `activity` interval of the raising-arms sensor. If the conductor takes too much time to trigger the next note, the timer expires and activates the singing actuator. In the case of *SingSong*, this mechanism generates a nice sensation of uninterrupted musical flow during the singing scene.

5.2 Interacting with SingSong

SingSong was designed considering both user and performer interaction. Figure 9, left, shows a user reacting to singer#1 complaints (just after it was commanded to stop chatting); the right side displays a miming clown conducting the chorus during the singing scene.

An analysis of the *SingSong* experience is beyond the scope of this paper. However, it is interesting to point out some observations we made during the runs of *SingSong* with users and performers, which, in our opinion, underscores our interest in immersive, story-based systems.

Users seemed to be quite comfortable in assuming the role of the conductor. In particular, they appeared to have a great time conducting the chorus. The simplicity of the interface coupled with the joy of generating interesting music provided a very pleasant experience. Also, the well-defined end to the interaction (signaled by the applauses) makes *SingSong* terminate just after a dramatic climax. These are precisely the kind of effects we should expect in story-based environments, in opposition to "exploring the world" immersive systems.

SingSong in the performance mode constitutes a typical experiment in *computer theater*, as defined in [5]. The choice of a clown costume, complete with red nose, produces an interesting effect: a more harmonious blending of real and the CG world. The performer's characterization as a clown somewhat puts him in a world as fantastic as the singer's virtual world.

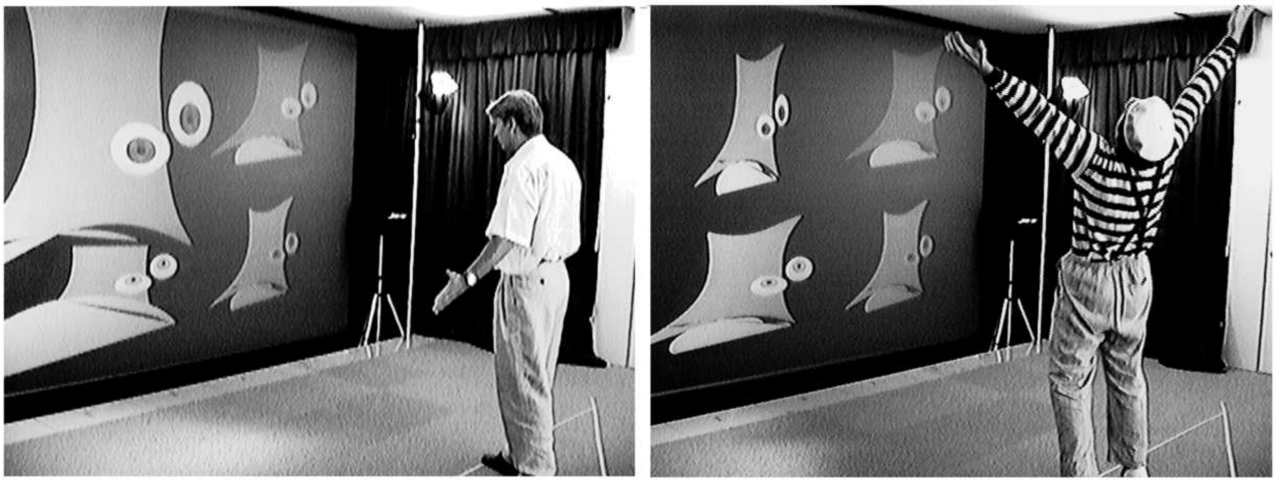

Figure 9: Two scenes from *SingSong*. The left picture shows a user playing with the system, while the right picture portrays a performance case.

6 Discussion

This research began with the observation that there are almost no paradigms and methodologies for scripting complex interactive systems. Especially in the case of stories with multiple, concurrent threads, current systems based on the reactive and on the tree-like paradigms seem to be insufficient and inadequate.

The interval script paradigm proposed in this paper is a first step towards more general tools and paradigms for interactive script writing. The method seems to have the required expressive capabilities, but, as the analysis of fig. 8 quickly reveals, it still lacks clarity and simplicity. Partial blame should be put in the task itself: it is very hard to describe and visualize multiple, sometimes unrelated activities.

But we do believe that the interval script paradigm, as described in this paper, still employs primitives which are too low-level: it is almost an "assembly" language for events in time. However, through our experience writing the interval script for *SingSong*, we have detected patterns of interval interconnection appearing many times in different situations in the script. Those patterns can be embodied in higher level externals defining, for example, chains of events, conditional branching, and loops. Nevertheless, a complete abstraction into those primitives may restrict the expressiveness of interval scripts to tree-like levels.

It is important to notice that immersive environments pose more difficult scripting problems than normal computer interfaces. The current interaction between computer and human is mostly based on non-gestural events (key typing, locator, selector). Both the sensing and the generation of gestural events require complex patterns of time interaction. In our paradigm, this was reflected in many instances, as, for example, in the need for `desired` and `actual` intervals for actuators.

On the other hand, story-based environments challenge event driven systems by requiring that past interaction to be taken into account. Meaning of gestures and actions must change as the action progresses. This is handled by the interval script paradigm by associating the same sensing routine with different externals (and thus, different intervals) in different scenes. As time progresses, the externals associated with the beginning of the script move into the past, turning off the sensing for the scenes which had already happened. Then, the progression of the story may reach the point where the externals of a new scene are activated, mapping the sensing routine into a new time interval, and therefore to a different context.

In some ways, we can see the interval mechanism as a way to switch the context of sensing and actuating routines. In *SingSong* we employed the same basic routines many times to produce completely different results in each scene: "arms up" means "stop chatting" in the chatting scene; later, it means "next note, please" in the singing scene.

The interval script structure also simplified the process of debugging the sensing and actuating routines, and the time structure itself. We designed general-purpose debugging routines which, embedded in externals, were activated according to specific conditions or in conjunction with the "problematic" externals. The interval script paradigm enables easy connection and disconnection of those debugging devices, in addition to provide a common structure which facilitates the design of general-purpose debugging routines.

We do not believe that it would be possible to implement *SingSong* as fast as we did without the interval script structure. The interval script provided a flexible method to change the script as we are designing new routines and testing the interaction. In spite of the low level of the language, and our lack of experience in thinking in terms of local relationships between intervals in time, the interval script paradigm considerably simplified the task.

With the experience provided by *SingSong*, we plan to implement higher level structures and externals to facilitate the understanding and reading of interval scripts. Among our future projects, we plan to use interval scripts to control an immersive, evolving diary describing a child's experience, as well as some further and more artistically ambitious experiments in computer theater.

References

[1] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] Masaaki Fukumoto, Kenji Mase, and Yasuhito Suenaga. Finger-pointer: Pointing interface by image processing. *Comput. & Graphics.*, 18(5):633–642, May 1994.

[3] Tinsley A. Galyean. *Narrative Guidance of Interactivity*. PhD thesis, M.I.T. Media Arts and Sciences Program, 1995.

[4] Pattie Maes, Trevor Darrell, Bruce Blumberg, and Alex Pentland. The ALIVE system: Full-body interaction with autonomous agents. In *Proc. of the Computer Animation '95 Conference*, Geneva, Switzerland, April 1995.

[5] Claudio S. Pinhanez. Computer theater. Technical Report 378, M.I.T. Media Laboratory Perceptual Computing Section, May 1996.

[6] Claudio S. Pinhanez and Aaron F. Bobick. PNF Calculus: Representing and propagating time constrains in Allen's interval algebra. Technical Report 389, M.I.T. Media Laboratory Perceptual Computing Section, September 1996.

[7] Naoko Tosa, Hideki Hashimoto, Kaoru Sezaki, Yasuharu Kunii, Toyotoshi Yamada, Kotaro Sabe, Ryosuke Nishino, Hiroshi Harashima, and Fumio Harashima. Network-based neuro-baby with robotic hand. In *Proc. of IJCAI'95 Workshop on Entertainment and AI/Alife*, Montreal, Canada, August 1995.

[8] Naoko Tosa and Ryohei Nakatsu. For interactive virtual drama: Body communication actor. In *Proc. of 7th International Symposium on Electronic Art*, Rotterdam, The Netherlands, September 1996.

[9] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 373–381. Morgan Kaufmann, San Mateo, California, 1990.

[10] Christopher R. Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. Technical Report 353, M.I.T. Media Laboratory Perceptual Computing Section, 1995.