Designing Story-Based Interfaces by Interval Scripts

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 This paper argues that interaction in immersive, story-based interfaces can be described by scripts based on temporal intervals and their local relationships. In such *interval scripts* the system's designer attaches sensor and actuator routines to time intervals and the development of the interaction is defined by local precedence relations provided by the designer of the system. Interval scripts have the potential to express more clearly the complex structure of time events in immersive environments where the user's actions span non-punctual intervals of time, making the use of event-based interaction design quite difficult. This paper examines in detail two scenes implemented in a prototype interactive system called *SingSong*.

インターバル・スクリプトを用いた物語的インタフェースの記述法

あらまし 臨場感があり物語的な展開のあるシステムのインタフェースを、時間の区間とその局所的な関係に基づくスクリプト（インターバル・スクリプトと呼ぶ）手法を用いて記述できることを示す。それは個々のセンサー関数とアクチュエータ関数を時間区間に結合させておいて、インタラクションの展開をその時間区間の並びを定義することによって記述する。本文では *SingSong* と呼ぶ実験システムの2つの場面をとりあげて、スクリプトの記述法を詳しく述べる。

1 Introduction

The design of human machine interaction requires a lot of knowledge and experience in many different fields. Scripting of the interaction is an important subject which has not been well discussed especially in the case of multi-modal and gesture-based scripted interactions ([2]). One of the objectives of our current research is the design and implementation of an interaction manager system which is able to handle complex patterns of immersive interaction evolving through time. The interaction manager should be able to track multiple concurrent stories turned on or off according to the development of a basic story and the users' actions. There have been few experiments with story-based immersive systems ([3]) and we are also interested in examining the impact that some dramatic features present in stories can have in interface design ([4]).

Also, gesture- and action-based interaction embodies physical actions which make interfaces fundamentally different to conventional monotonic interfaces based on keyboard, mouse, or even speech. We want to be able to step further from the "ask & tell" interface era to the "action & feel" era where users become able to use their whole body for interaction in immersive environments ([5, 6, 8]). To handle the design of such immersive environments it is necessary to develop scripting tools which abstract sensor and actuator control issues and enable direct expression of interactive action.

In this paper we propose a novel script paradigm for immersive, interactive systems based on the concept of *interval scripts*. Scripting using intervals is discussed and exemplified in an interactive system named *SingSong* which was designed and implemented using the paradigm. We have noticed that users seemed to be quite comfortable to interact with the active objects in the *SingSong* environment mostly due to the loose constraints on duration and timing of action which is a main feature of interval scripts.

For the actual run of the system we developed an interaction manager which uses the interval script to determine when it is appropriate to call sensing and actuating routines.

The interaction manager was developed based on the PNF calculus of Pinhanez and Bobick ([7]). This paper also provides detailed examples of the interval script of two scenes and a general evaluation of the strengths and weaknesses of the paradigm.

2 A Novel Paradigm: 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 temporal intervals corresponding to the different actions and events and the temporal 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.

2.1 Allen's Interval Algebra

To model the time relationships between two intervals we employ the interval-based time representation proposed by Allen [1]. The representation is based on the 13 possible primitive relationships between two intervals which are summarized in fig. 1.

Given two events in the real world, their possible time relationship can always be described by a disjunction of the primitive time relationships. For instance, the action of driving a car must happen while the car engine is on. Using Allen's temporal primitives, the interval associated with driving the car either STARTs or FINISHes or happens DURING or is EQUAL to the interval when the engine is on. That is, the time relationship between driving and having the 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 it is

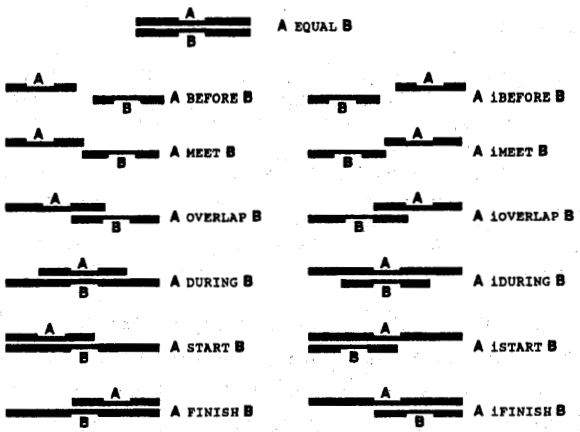


Figure 1: Allen's Interval Algebra: The possible 13 primitive time relationships between 2 intervals [1].

declared that 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.

To use an interval script in the real-time control of an interactive system we developed an interaction manager program based on the PNF calculus of Pinhanez and Bobick ([7]). The interaction manager is responsible for controlling all sensor and actuator routines, deciding when it is appropriate to call each routine and transferring values from sensors to actuators. The manager stores the interval script in an internal table and is able to maintain information about events which happened in the past.

2.2 Connecting to the Real World

According to our proposal of interval scripts the interaction of a system is described by temporal intervals connected to sensors and 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 dif-

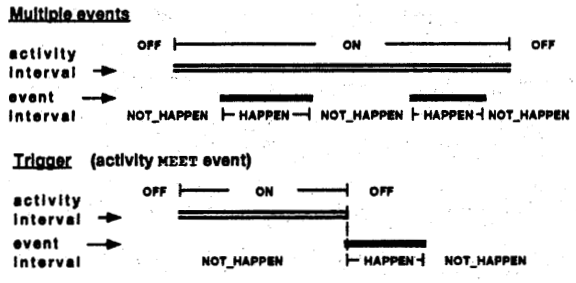


Figure 2: Intervals associated with a sensor, in two different configurations.

ferent sensors and actuators. In fact, an external seamlessly encapsulates the connections between a designer's routine and the interval structure used to manage the interaction. Quite commonly more than one interval is associated to one external as shown later.

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 an *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 three values: `HAPPENING`, `NOT-HAPPENING`, or `UNKNOWN`. During the time the activation interval is or may be happening, the interaction manager sends an `ON` command to the designer's routine, and `OFF` otherwise.

Figure 2 shows the intervals associated with a sensor in two different situations. In the first situation it is shown that the **event** interval can occur many times while the **activity** interval is in `ON`. The second situation shown in the bottom of fig. 2 exemplifies 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`s or happens `BEFORE` the **event** interval.

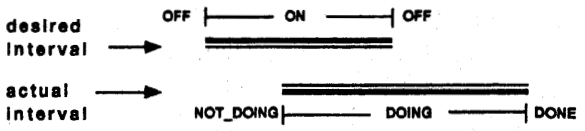


Figure 3: Intervals associated with an actuator.

Actuators

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

Those characteristics of actuators are reflected in our system by the association of two intervals to actuators: a *desired* interval and an *actual* interval. Figure 3 shows the relationship between the intervals and the actuator routine provided by the designer. When the **desired** interval is happening, the interaction manager sends `ON` messages to the designer's routine. When, and if the actuator goes on, the returning `DOING` message tells the interaction manager that the action is really occurring and therefore that the **actual** interval is happening.

When the **desired** interval finishes, 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.

Timers

Although the general objective of this proposal is to write scripts 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

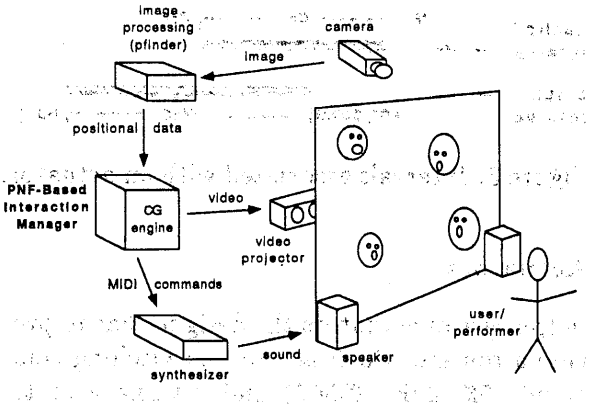


Figure 4: The basic setup of *SingSong*.

to turn the timer on and off; the `actual` interval — especially its end — can be used to trigger other actions as the timer expires.

3 Designing an Application with Interval Scripts

The methodology and algorithms described in this paper have been tested in a story-based, interactive system named *SingSong*. Figure 4 shows the basic structure of *SingSong* where a large video screen displays the image of four computer graphics-animated characters which can "sing" musical notes (as produced by a synthesizer). A camera watches the user or performer determining the position of her head, hands, and feet. The body position is recovered by the software *pfinder* developed at the MIT Media Laboratory ([10]).

SingSong makes the user take the role of a conductor of a chorus of four computer-graphics characters. The story starts with the "singers" animatedly talking to each other. The conductor enters and, to get the singers' attention, raises her arms. This gesture stops the chatting of only three of the singers, since one of them — singer#1 — does not stop until being commanded twice and after complaining by uttering sounds and increasing its size.

The next step for the conductor is to tune the singers. A tuning fork appears on the screen and the conductor has to point to one particular singer and simulate hitting the fork by waving her arm. In that case the tuning fork produces a musical note which is immediately repeated by the singer. However, singer#1 continues to misbehave and does not produce the right note unless the conductor knees down and pleads. After all singers are tuned the conductor can start the song by raising her arms. Each new note of the song is triggered by a raising of the arms. When the song is finished, applauses are heard, and the singers follow the conductor if she bows back. When the conductor starts to leave the scene, singer#1 strikes a new round of complaints.

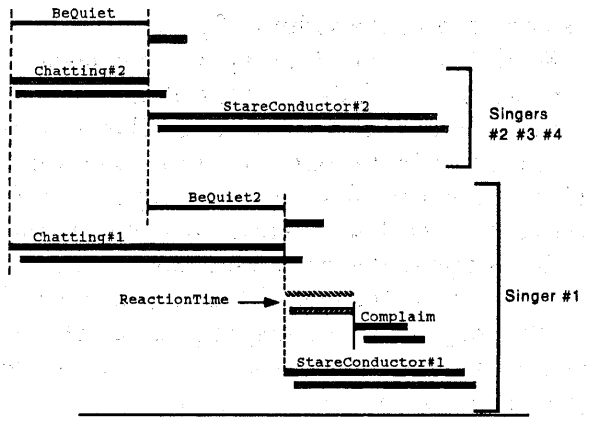
All the interaction is nonverbal, the user or performer gestures and the CG-characters sing notes and move. As we see, an important feature of *SingSong* is the fact that it immerses the user in a simple story which unfolds as the interaction proceeds.

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 [6], our system provides computer-generated partners to a human performer which are not only reactive but also able to follow a pre-defined script. 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 the scene from outside because she can clearly react to the situations and expressively displays her emotions.

3.1 Interval Script of the First Scene

The first scene of *SingSong* employs several different externals. `Chatting` is an actuator which controls the sound and the mouth movements simulating the chatting action of each singer. Four copies of `Chatting` are used, one for each singer. `BeQuiet` and `BeQuiet2` are trigger sensors which fire if the user raises both arms above his head. `StareConductor` is an actuator which makes a singer's eyes follow the conductor around the space. To provide a short pause between the conductor's gesture and singer#1's complaints we define a 3-second timer called `ReactionTime`. Finally, `Complain` is an actuator — used solely by singer #1 — which controls the sounds and graphics related to the complaining action.

Figure 5 displays a diagram showing how the different intervals of each external are temporally related. The diagram shows the relationships for singer#1 and singer#2, and the re-



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

Figure 5: Diagram of temporal relations and interval script corresponding to the first scene of *SingSong*.

lationships for the other two singers are identical to those of #2. Figure 5 also shows the interval script corresponding to the first scene (where the details of the C++ code are omitted for clarity).

The interval script describes completely how each external relate to the others. Initially the **desired** interval of all **Chatting** actuators and the **activity** interval of **BeQuiet** are defined to start together. This is shown in the first part of the script in fig. 5 which states that the **activity** interval of the sensor **BeQuiet** must START or EQUAL or iSTART the **desired** interval of **Chatting#1** and **Chatting#2**. In the temporal diagram of fig. 5 we represent this definition by the dashed line joining the beginning of both intervals.

The next definition states that the **event** interval of **BeQuiet** — **BeQuiet.event** — finishes the **desired** interval of **Chatting#2**, i.e., the singers are commanded to stop chatting when the conductor raises his arms. **BeQuiet.event** 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. 5.

However, since singer#1 does not stop chatting until 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** triggers — START or EQUAL or iSTART — **BeQuiet2.activity**.

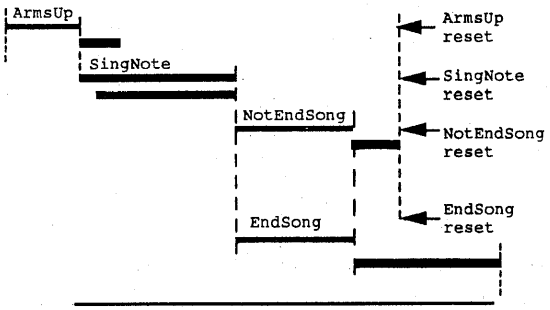
A detection of an **event** by **BeQuiet2** shuts off the sensor's activity, starts the **StareConductor#1**, and the **desired** interval of timer **ReactionTime**. The end of **ReactionTime.actual** starts the **Complain** actuator, finishing the first scene.

As it can be seen in fig. 5, 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.

3.2 Interval Script of the Singing Scene

The singing scene of *SingSong* is basically a loop of short interactions between the singers and conductor who triggers the next note of a song by raising his arms. Figure 6 shows the temporal diagram of the singing scene and its defining script.

The singing of each note starts with the activation of the sensor **ArmsUp** which fires only when both the conductor's arms are up. When **ArmsUp.event** occurs, its corresponding sensor is turned off and the actuator **SingNote** is started as described in the first two statements of the interval script of fig. 6. It is important to notice that actuators are always "started" through the **desired** interval. The beginning of the **actual** interval is completely determined by the output of its corresponding actuating



ArmsUp activity	MEET BEFORE	ArmsUp event
ArmsUp event	START EQUAL ISTART	SingNote desired
SingNote actual	FINISH EQUAL IFINISH	SingNote desired
SingNote actual	MEET	NotEndSong activity
SingNote actual	MEET	EndSong activity
EndSong activity	MEET BEFORE	EndSong event
NotEndSong activity	MEET BEFORE	EndSong event
EndSong activity	MEET BEFORE	NotEndSong event
NotEndSong activity	MEET BEFORE	NotEndSong event
NotEndSong event	MEET BEFORE	ArmsUp reset
NotEndSong event	MEET BEFORE	SingNote reset
NotEndSong event	MEET BEFORE	NotEndSong reset
NotEndSong event	MEET BEFORE	EndSong reset

Figure 6: Diagram of temporal relations and interval script corresponding to the singing scene of *SingSong*.

routine.

Therefore, while **SingNote.desired** is happening, **ON** commands are being sent to the routine which produces the next note of the song but the **actual** interval happens only when the routine successfully manages to send a MIDI **NoteOn** command to the synthesizer. The interval **SingNote.actual** ends when the synthesizer finishes producing the musical note, triggering the end of **SingNote.desired** and the start of two sensors, **NotEndSong** and **EndSong** as described in the script of fig. 6.

NotEndSong and **EndSong** are complementary sensors which detect if the all the notes of the song have already been sung. They constitute the terminate condition of the loop which plays the entire song. After being activated, the sensors check an internal, common variable which is updated every time a note is sung by the singers. If the end of the song is reached an **EndSong.event** occurs, triggering the end of both sensors and the start of the next scene (not shown in the script of fig. 6).

In the case that the song is not finished, **NotEndSong.event** shuts off both sensors and triggers a special interval associated with every external called the **reset** interval. Whenever a **reset** interval happens any intervals associated with that particular external are set to an undetermined state even if the intervals had already occurred. For example, if the **reset** interval of a sensor happens both the **activity**, the **event**, and the RESET interval itself are set to an undetermined state. Basically this enables a movement backwards in time which, when coupled with a sensor of loop termination, makes the construction of loops possible.

The 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.

3.3 Interacting with SingSong

SingSong was designed considering both user and performer interaction. The left picture of fig. 7 shows a user reacting to singer#1 complaints (just after it was commanded to stop chatting); the right picture 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 seems to be quite comfortable in assuming the role of the conductor. In particular they appear to have a great time conducting the chorus. The simplicity of the interface cou-

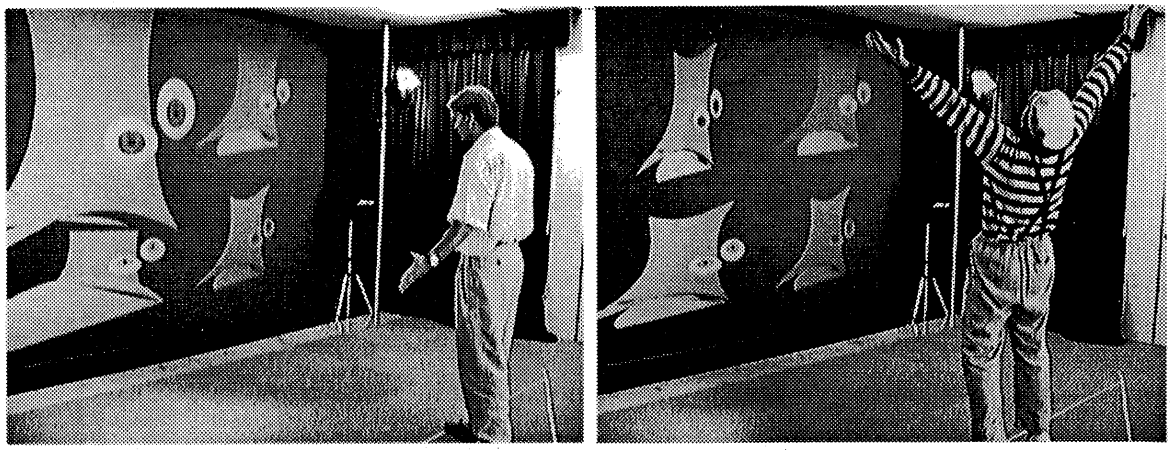


Figure 7: Two scenes from *SingSong*. The left picture shows a user playing with the system, while the right picture portrays a performance case.

pled with the joy of generating interesting music provides 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 effect we want in story-based environments.

SingSong in performance mode constitutes a typical experiment in *computer theater* as defined in [6]. The choice of a clown costume, complete with red nose, seems to produce an interesting effect of blending the 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.

4 Discussion

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 figures 5 and 6 quickly reveals, it still lacks clarity and simplicity. Partial blame should be put in the task itself since it is very hard to describe and visualize multiple, sometimes unrelated activities.

We see several reasons to use Allen's algebra to design interactive scripts based on temporal intervals:

- No explicit mention of the interval duration or specification of relations between the interval's extremities is required.
- The existence of Allen's algorithm for propagation of time constraints allows the designer to declare only the relevant relations leading to a cleaner script.
- 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.

We believe that the interval script paradigm, as described in this paper, still employs primitives which are too low-level. Interval scripts look like an "assembly" language for events in time. However, throughout our experience writing the interval script for *SingSong*, we have detected patterns of interval interconnection which appear 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.

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

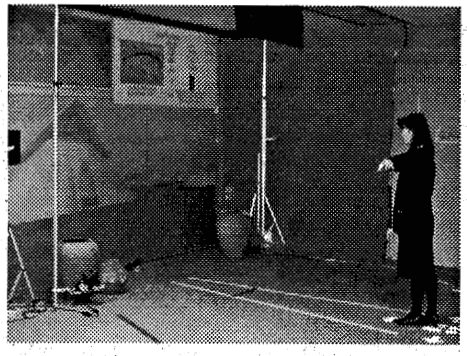


Figure 8: An immersive, interactive environment portraying a visit to an ancient Yayoi village.

the generation of gestural events require more 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.

Currently we are also applying the interval script paradigm for an immersive, interactive virtual reality environment where users can walk through a virtual space and explore the hyper-linked information space. Unlike the conventional systems which use the same event handling scheme for essentially different input media, i.e. body gestures, mouse dragging, or keyboard strokes, the interval script will be able to provide relaxed, multi-modal interactive environment. Figure 8 shows the test VR environment which enables the user to walk through an ancient Japanese village ([5]). The system was originally developed using a conventional event handling mechanism. We are starting to re-code the interaction in this system using interval scripts both as a way to improve the interface and as a test for the interval script paradigm.

Designing and implementing *SingSong* as fast as we did would not be possible without using the interval script paradigm. 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 the interval script paradigm considerably simplified the task.

Acknowledgments

The work of Claudio Pinhanez in the *SingSong* project was supported by a grant from the Starr Foundation through the MIT-Japan program.

References

[1] ALLEN, J. F. Towards a general theory of action and time. *Artificial Intelligence 23* (1984), 123–154.

[2] FUKUMOTO, M., MASE, K., AND SUENAGA, Y. Finger-pointer: Pointing interface by image processing. *Comput. & Graphics. 18*, 5 (May 1994), 633–642.

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

[4] LAUREL, B. *Computers as Theatre.* Addison-Wesley, Reading, Massachusetts, 1991.

[5] MASE, K., KADOBAYASHI, R., AND R. Metamuseum: A supportive augumented reality environment for knowledge sharing. In *Intn'l Conf on Virtual Systems and Multimedia'96* (Sept. 1996).

[6] PINHANEZ, C. S. Computer theater. Technical Report 378, M.I.T. Media Laboratory Perceptual Computing Section, May 1996.

[7] PINHANEZ, C. S., AND BOBICK, A. F. PNF Calculus: A method for the representation and fast recognition of temporal structure. Technical Report 389, M.I.T. Media Laboratory Perceptual Computing Section, Sept. 1996.

[8] TOSA, N., AND NAKATSU, R. For interactive virtual drama: Body communication actor. In *Proc. of 7th International Symposium on Electronic Art* (Rotterdam, The Netherlands, Sept. 1996).

[9] VILAIN, M., KAUTZ, H., AND VAN BEEK, P. Constraint propagation algorithms for temporal reasoning: A revised report. In *Readings in Qualitative Reasoning About Physical Systems*, D. S. Weld and J. de Kleer, Eds. Morgan Kaufmann, San Mateo, California, 1990, pp. 373–381.

[10] WREN, C. R., AZARBAYEJANI, A., DARRELL, T., AND PENTLAND, A. Pfinder: Real-time tracking of the human body. Technical Report 353, M.I.T. Media Laboratory Perceptual Computing Section, 1995.