Flexible Timeline User Interface using Constraints

Kazutaka Kurihara
Department of Computer Science
The University of Tokyo
7-3-1 Hongo, Bunkyoku Tokyo,
1130033, Japan
qurihara@ui.is.s.u-tokyo.ac.jp

David Vronay
Center for Interaction Design
Microsoft Research Asia
3F, Beijing Sigma Center No.49
Zhichun Road Haidian District
Beijing 100080, P.R. China
davevr@microsoft.com

Takeo Igarashi
Department of Computer Science
The University of Tokyo
/ JST PRESTO
7-3-1 Hongo, Bunkyoku Tokyo,
1130033, Japan.
takeo@acm.org

Abstract
Authoring tools routinely include a timeline representation to allow the author to specify the sequence of animations and interactions. However, traditional static timelines are best suited for static, linear sequences (such MIDI sequencers) and do not lend themselves to interactive content. This forces authors to supplement their timelines with scripted actions which are not represented. Timelines also force frame-accuracy on the author, which interferes with rapid exploration of different designs. We present a redesign of the timeline in which users can specify the relative ordering and causality of events without specifying exact times or durations. This effectively enables users to "work rough" in time. We then implement a prototype and perform a user study to investigate its efficiency.

Categories & Subject Descriptors: H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces.

General Terms: Design.

Keywords: Prototyping; Timeline; Constraint Solving.

INTRODUCTION

Software for working with temporal data, such as music or animation, often employs a timeline as its central representation for sequencing and synchronizing events. For the straightforward display of a small number of channels of sequential events, this representation works quite well.

However, this metaphor which was originally designed for straightforward animation is now being used to design complex interactive applications. Under these conditions, the timeline breaks down.

Our team was tasked with designing the user interface for a new authoring tool targeted at rapid application prototyping. Having experienced the limitation of current tools in our own prototyping work, we set out to understand the shortcomings of the different representations and design an alternative. Other teams were already working on the programming language and graphics system, and so the redesign of the timeline UI became our central focus.

CHI 2005, April 2–7, 2004, Portland, Oregon, USA.
ACM 1-59593-002-7/05/0004.

Timeline Limitations

To understand the limitations, we interviewed six professional interaction designers who were self-described experts with either Flash[1] or Director[2] or both. Four of the subjects worked within our own company and two were unaffiliated. Subjects were not compensated for their participation beyond a refund of transportation costs and lunch. We combined the results of these interviews with our own personal observations. We found that the timeline metaphor is very easy for subjects to understand and begin working with. The grid of frames makes it very easy to specify the exact duration, sequence, and synchronization of animation events. They can easily see exactly what is on the screen at a given instant in time.

However, the timeline breaks down when asked to handle the needs of interactive applications. There are three general weaknesses: First and most obvious is that the timeline is a line -- it is a linear representation, and interactions are inherently non-linear. When it comes to the representations of loops and conditions, the timeline cannot help. The designer is forced to jump around the timeline, sticking sequences where they might fit and relying on embedded code to control the playback of time.

The second weakness is that the timeline is global, always showing the entire application. Combined with the sequential display, this results in the timeline quickly becoming inefficient for large or complicated projects. As we saw in point 1 above, interactive sequences force the user to arbitrarily cut up the timeline, inserting sequences wherever they might fit and controlling the execution of them with embedded jumps. As timelines stretch to tens of thousands of frames, subjects forget where in time they placed different animated sequences. This confusion also exists in the vertical dimension. With potentially hundreds or even thousands of objects in a complex application, there is no straightforward way of knowing which channel contains a particular object at a particular time. The author just has to remember.

The third and final weakness is that the timeline enforces a very literal and exact notion of time on the author. There is no provision for specifying things loosely. The timeline requires you to say things like "at 5:00, do this for exactly 13.5 seconds". You cannot say loose expressions like "do this after that" or "these have to last the same amount of

time, but I don't know how long that will be." The inability to "sketch" in time was a consistent complaint.

This exacting notion of time prevents the user from "working rough". In [3], Wong discusses the importance of working at a rough level during the initial exploratory phases of design. Storyboarding is a very common technique in the design of both interactive applications and linear movies. Tools like Denim[4] allow a designer to quickly and easily sketch out both the user interface and simple causality. The current timeline makes such exploration and sketching extremely difficult. The user trying different variations is constantly wrestling with the timeline, inserting and deleting frames, pushing things around, and generally rearranging them to fit his needs. As the application develops and the timeline becomes more complex and fragmented, the author is increasingly discouraged from messing with it in the fear that something will break.

The inability to specify actions in a loose and relative way also makes many tasks overly complicated. Consider the common task of playing an animation while the system is processing some data. The literal timeline provides no way to do this. The author instead has to create an animation loop of a fixed duration and control it programmatically.

These three weaknesses taken together render the current timeline non-optimal for interactive multimedia authoring.

Related Work

Flash MX[1] supports the notion of hierarchical channels in its timeline, where one layer can contain other layers. These other layers have their own clock and can be hidden when not used. This makes it very easy to script relative animations, such as a moon that orbits a planet while the planet orbits the sun. While this helps somewhat to control clutter, it does not address the more serious problem of hiding the interaction.

Wolber in [6] extends the timeline metaphor to support multiple synchronous streams. While this makes the authoring of simultaneous tasks easier and reduces timeline clutter, it still does not address the difficulty of viewing interaction.

TIMELINE REDESIGN

The goal of our redesign was to preserve all of the positive aspects of the old timeline while overcoming all of the weaknesses. Our current design comes a long way towards that goal, while also providing a number of new powerful capabilities.

Key to achieve our goal was to support a "sketch" experience to the authoring task. By sketch, we do not mean literally sketching with a pen, but rather the general attitude that the user need only specifies the information he or she desires. The rest is inferred by the temporal constraint solver. This gives users the maximum ability to explore, create variations, make adjustments, and change their minds.

Design Concepts

Events happen in time -- our core metaphor is that of events occurring in time. An event can be any sort of verb, from an object being created to a property being changed to everything on the screen turning blue. Events can also just be simple text labels with no functionality, just serving as comments, placeholders, or temporal reference points.

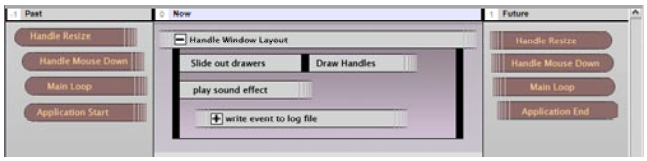

Figure 1: Timeline as a Query Viewer

Time is relative -- rather than a single uniform linear timeline, we divide time into three broad sections -- past, present, and future (Figure 1 left, center, and right). The present area contains the subset of the application the user is currently working on, while the past and future areas contain simplified representations of events that happen before or after the present action. The timeline, then, can be thought of as a query viewer (Figure 1 overall). Rather than showing every event in the entire system, the timeline normally displays a subset of the events that match a particular query. Given any set of events, like "every event involving a button on the main window", the timeline can determine their execution order and containment relationships and render them to the screen. In this way, the user can easily scope the view to just those events he or she currently cares about.

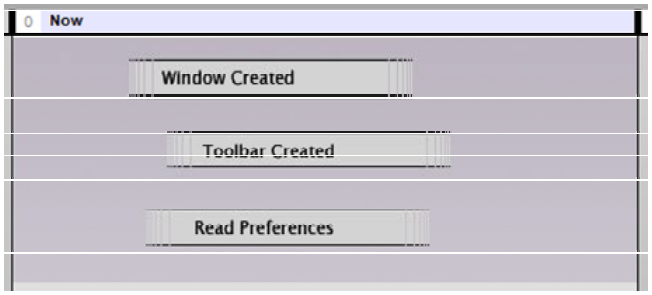

Figure 2: Three Events

Events can be fuzzy -- when creating an event, the user does not have to specify any information about its time at all. For instance, in Figure 2 below, we can see three events. The user has simply said "these three things happen", and has not said anything about their relative orders or durations. The fuzzy edges indicate an unspecified time (for clarity, we will use text label events in these images).

Events can also be specific -- if the user desires, the start, end, and duration of events can be fixed. In Figure 3, we see the user has said the tool bar is created after we begin window creation, and that the preferences are read after the toolbar creation has completed.

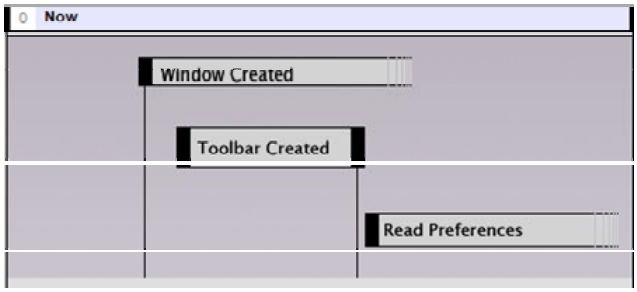

Figure 3: Specific Events

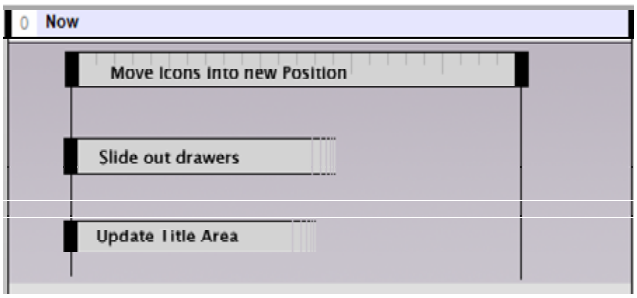

Figure 4: Specific and Non-specific Durations Mixed

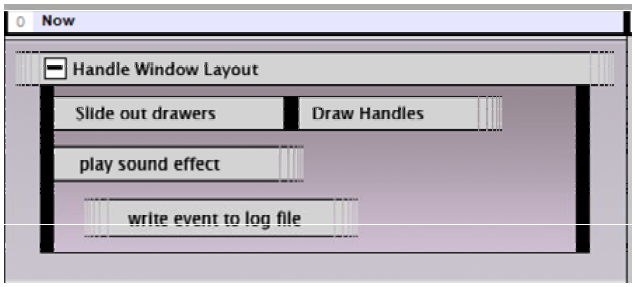

Figure 5: Hierarchical Timelines

The hard bold edges are used to indicate specific times. The long vertical lines are guides that can be shown for emphasis or hidden at the user's request.

Thus, the user is really specifying temporal constraints rather than exact literal time. Only the constraints the user cares about (such as sequencing) are defined, and all other actions (such as the duration of these events or the end time of the window creation event) are left unconstrained. Drapeau[5] and others have previously observed the importance of constraints in multimedia authoring.

Of course, the user can also specify exact durations, and can even work with a combination of the two. In Figure 4, the icon animation of exactly 16 frames is sync with two other animations whose duration is not specified.

Events can contain their own timeline -- the timeline is hierarchical and events themselves can have timelines inside of them. Within an event, sub-events can always get a specific temporal reference to the start and end times of their parent, even the times of the parent are unspecified. In Figure 5, we can see that the Handle Window Layout event has been expanded to reveal four other events inside. At the beginning of this event, the drawer slide-out animation and the sound effect start at the same time. Once the drawers are done sliding out, the handles are drawn. Also, at some point, an event is written to an event log. All of these events must be completed before the Handle Window Layout event is considered done.

With its support of hierarchy, our timeline lends itself to going from a quick sketch or storyboard directly into the finished product. It also supports designers and programmers working together. A designer can lay down the larger blocks and specify causality, sequence, and timing. Then, without loosing any work, the programmer can add in the details by adding code to the blocks. Highly specified code can co-exist with very general sketchy behavior, and whole chunks of interactions can be dragged and dropped from place to place.

With its support of hierarchy and relative time, the timeline also allows non-linear features such as loops, conditionals and other control blocks.

IMPLEMENTATION

The improved timeline we have discussed was originally designed for a new authoring tool called AfterThought which is specifically designed to support rapid prototyping of interactive applications. We implemented a prototype of our new timeline interfaces using C#.NET. It can run on today's common computers. It has functions to create events and edit constraints using clicks and drags. Nested timelines are also supported.

The temporal relations among events in the timeline are automatically handled by a constraint solver. There are number of well-known constraint solvers such as Borning's [7] (constraint hierarchy) and Gleicher's [8] (solving dynamics), and Yamane's [9](also solving dynamics). We currently use Yamane's method but any other solvers would work as well.

USER STUDY

We performed a study to investigate the usability of the prototype timeline user interface for AfterThought, comparing it with Macromedia Flash MX and Adobe Premiere Pro[10]. Eighteen college students who had previous expertise with Flash or Premiere or both participated in the study.

Procedure and Design

First, we instructed them how to use the three tools for editing timelines. Once they were comfortable, we showed them 3 timelines for their tasks (Figure 6). They were told to insert 5 new events at a point for each timeline (shown by "Here" in Figure 6) using the 3 tools. The orders in which each participant used the tools were counterbalanced. An important point was that they had to maintain all the predefined constraints such as "These events end at the same time" (represented by dotted lines in Figure 6). The initial timeline was constructed in each tool. However, the partici-

pants were required to set proper causality constraints on the events they newly inserted. After they finished each task, they evaluated how easily they were able to achieve the task by choosing a number from 1 (most difficult) to 5 (easiest). We also measured the consuming times for all the tasks. However, it was not very useful because it varied too much depending on each participant's skill and experience.

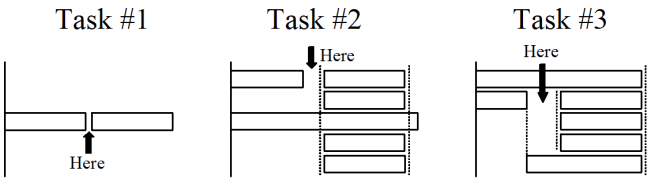

Figure 6: Timelines for Task #1, Task #2, and Task #3

Results and Findings

Figure 7 shows the averages of the evaluations for each task and each tool. A Scheffe multiple-comparison test showed that AfterThought scored significantly better than the other 2 tools did in task #1 ($p=0.036$ against Premiere, $p=0.003$ against Flash) and task #2 ($p=0.045$ against Premiere, $p=0.000$ against Flash). We observed that the participants had difficulty in recovering the predefined constraints they broke when they inserted new events in Flash and Premiere. It was not the case with AfterThought, which preserved all the predefined constraints automatically.

Meanwhile, one-way analysis of variance showed that there was no significant difference among 3 tools in only task #3 ($F_{2,51}=2.25$, $p=0.115$). Our observations indicated that this was due to the lack of some needed functionality in our timeline design. Many participants commented that it was that there was not enough visual feedback for indicating what was and was not constrained.

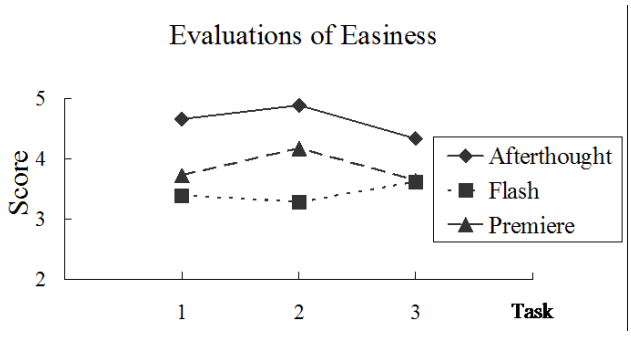

Figure 7: Evaluations of Easiness

Improvements

Based on our results, we revised the timeline design. The next version will feature:

- More visualization for existing constraints.
- A mode or a shortcut button for explicit invalidation of all the existing constraints.
- Implicit relaxation of constraints when the user wishes to change the situation.

CONCLUSION

In this paper, we proposed a new "relative" timeline and user interfaces for it, considering the existing problems of timeline editing for multimedia authoring. We implemented a prototype and performed a user study that supported their effectiveness and versatility.

User interfaces involving constraints generally have two problems. One is how to set or remove constraints easily, and the other is how to show the user active constraints. Our interfaces solved these problems by hiring simple interaction of "clicking on the edges of events" and its visualization. We observed that it works well when the number of events is relatively small. Our future work is to investigate other interfaces and visualizations by which the user can deal with complicated constraints.

ACKNOWLEDGMENTS

The authors would like to thank Harry Shum for management support, Alexander Stojanovic and Matthew MacLaurin for AfterThought development, Shuo Wang, Fei Zhang and Jingbo Dong for assistance with user studies and prototyping, and Hiroshi Hosobe for technical comments.

REFERENCES

1. Flash MX, software by Macromedia, Inc.
2. Director, software by Macromedia, Inc.
3. Wong, Y.Y. Rough and ready prototypes: Lessons from graphic design. *In Short Talks Proceedings of CHI '92: Human Factors in Computing Systems*, Monterey, CA, May 1992, pp. 83-84.
4. Mark W. Newman, James Lin, Jason I. Hong, and James A. Landay, "DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice." *In Human-Computer Interaction*, 2003. 18(3): pp. 259-324.
5. Drapeau, George D. Synchronization in the MAEstro multimedia authoring environment, *Proc. First ACM international conference on Multimedia*, 1993, 331-339.
6. Wolber, D. A Multiple Timeline Editor for Developing Multi-Threaded Animated Interfaces. *UIST 1998*, ACM Press, 117-118.
7. Borning, A., Marriott, K., Stuckey, P., and Xiao, Y.:Solving Linear Arithmetic Constraints for User Interface Applications, *Proc. ACM UIST, 1997*, pp.87-96.
8. Gleicher, M.: A Differential Approach to Graphical Interaction, Technical Report CMU-CS-94-217, School of Computer Science, Carnegie Mellon University, 1994.
9. Katsu YAMANE and Yoshihiko NAKAMURA: " Synergetic CG Choreography through Constraining and Deconstraining at Will," *Proc. IEEE ICRA2002*, Vol.1, pp.855-862, Washington D.C., U.S.A., May, 2002.
10. Premiere Pro, software by Adobe Systems, Inc.