SEMANTICS OF TIME TRAVEL IN A GENERATIVE INFORMATION SPACE

A Thesis

by

MADHUR J. KHANDELWAL

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2004

Major Subject: Computer Science

SEMANTICS OF TIME TRAVEL IN A GENERATIVE INFORMATION SPACE

A Thesis

by

MADHUR J. KHANDELWAL

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Andruid Kerne (Chair of Committee)	Frank Shipman (Member)
Carol LaFayette (Member)	Valerie E. Taylor (Head of Department)

December 2004

Major Subject: Computer Science

ABSTRACT

Semantics of Time Travel in a Generative Information Space. (December 2004)

Madhur J. Khandelwal, B.E., Birla Institute of Technology and Science, Pilani, India

Chair of Advisory Committee: Dr. Andruid Kerne

This thesis focuses on interactive and computational semantics for manipulating the time-based medium of an evolving information space. The interactive semantics enable the user to engage in linear timeline traversal and non-linear history manipulation. Extended tape recorder metaphor controls, including jog-shuttle based navigation, provide the user with flexible means for operating the software's generative functionalities, and linearly traversing session history. The user can see previews of information space states while traversing the history using the jog-shuttle. We also introduce a door-latch metaphor that enables one of several considered forms of non-linear history manipulation. Users can change history by retroactively latching an information sample in its position across time.

For representing the information space history, we have developed MPEG-like computational keyframe semantics. This representation is in the form of XML, which is generated automatically and converted back to Java by a framework named ecologylab.xml, which was developed as a part of this thesis. These computational keyframe semantics serve as the basis for interaction semantics. A user study was conducted in the form of a design competition, to evaluate these new features. The

results indicated that the users do find the time travel features useful and they feel more in-control of the information space with access to time travel features compared to the case when time travel features are not present.

To my parents

ACKNOWLEDGMENTS

I am indebted to my advisor, Dr. Andruid Kerne, for his unending support, guidance and encouragement in completing this thesis. I have learned enormously from him through the several interesting discussions and development sessions we have had during this research. His faith in my abilities to tackle this thesis has given me tremendous confidence during difficult times.

I would like to express my sincere thanks to Carol LaFayette, MFA, who was extremely interested in the work throughout. She assisted me greatly in designing the user study, and in recruiting subjects. I am also grateful to Dr. Frank Shipman for his insight and valuable suggestions on the work.

I have enjoyed working with the members of Interface Ecology Lab; all their support and comments on this work are appreciated. Special thanks to Michael Mistrot and Vikram Sundaram for helping me with the design and implementation of interactive non-linear history mechanisms. Finally many individuals, who shall remain anonymous, volunteered as subjects for the experiment. Without them, there would not have been a thesis.

Most importantly, I would like to thank my family and friends for their constant encouragement, love and belief in me throughout my graduate studies.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

1 INTRODUCTION

The combinFormation software provides a generative information space for browsing, collecting and organizing information from the Web [Kerne and Interface Ecology Lab 2004; Kerne 2001; Kerne and Sundaram 2003]. Composition is used to represent collections of information samples. The program supports mixed-initiative [Horvitz 1999] composition. That is, a composition of blended information samples develops over time as a result of direct manipulation by the user, and proactive initiative by a generative software agent. The current version of the software supports information samples in the form of text chunks and images. Each information sample serves as a navigational and semiotic surrogate [Burke 1999] for the web page it comes from and is hyperlinked to.

The agent's generative activities include procedural information collecting and procedural visual composition. By procedural information collecting, we mean that the program proactively crawls the web, retrieves relevant information and forms a collection of information samples. As a part of this, the program forms a structural model of how these samples are related to each other. Some of these information samples are composed into an evolving visual composition, which appears on the screen in the form of a streaming collage. Visual composition includes image-processing effects such as blur, desaturation and alpha blending of information samples. The user can interact with the information samples on screen using two kinds of operations.

This thesis follows the style of the *Journal of the ACM*.

The composition operations include moving the information samples spatially on screen, resizing them, changing the text-image mix, changing the rate at which the information samples are brought onto the screen and removing information samples from screen. The interest-expression operations include expressing positive or negative interest in the information samples and initiating searches to find related information. Users can combine each of the composition operations with interest expression modifiers, such as expressing positive or negative interest in a sample. As a result of interest-expression user interactions, the program forms a model of the user, in conjunction with its model of the information structure. This model serves as the basis for the program's generative processes of procedural information collecting and visual composition. A feedback loop is created from the model, to the visual composition, through the user's cognition and expressive interaction, and back to the model as seen in Figure 1. Computational representations, visual representations, and human cognition are connected through this loop. This enables the program to learn about the user's evolving interests. It gives the user influence on what the program does.

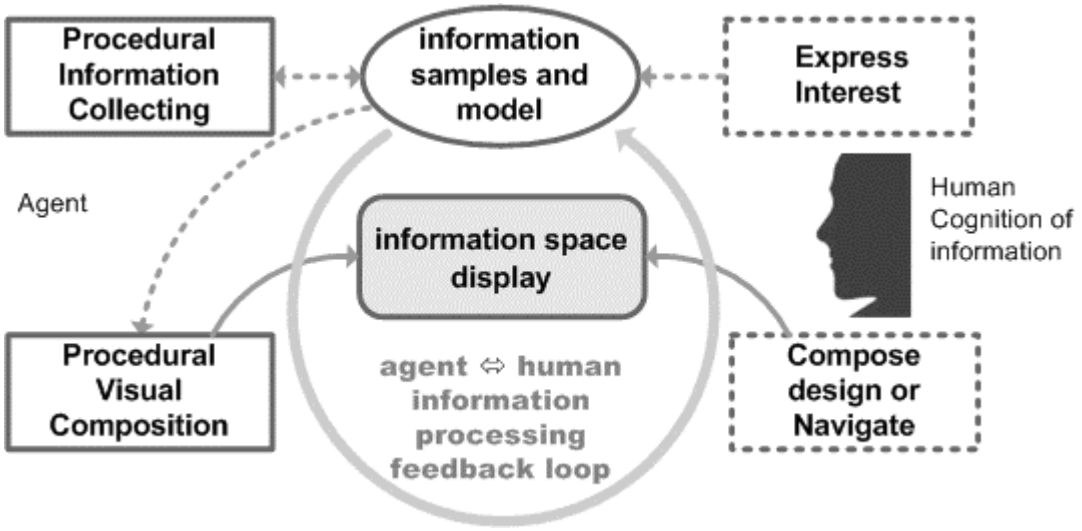

FIG. 1. Feedback Loop of Digital and Human Information Processing.
Model → Information Space → User → Model.

As a result of the combined activities of the user and the program, the state of the information space changes over time. When the state of the information space changes, its meaning changes, both for the user and the program. This may stimulate the user to have new ideas about what is important and how things are related [Kerne and Smith 2004]. Here, we consider representations of the information space. At any point in time, the following components define the semantics of the information space state:

- The visual appearance of the information space.
- The information pool, in the form of media collected by the program.
- The interests of the user and program's model of those interests. This could involve complete changes in topic and smaller focusing or widening of previous topics of interest.

- The user's cognition of the composition and its meaning, and her experience thereof.

Both the user and the program make changes in the state of the information space over time. Through this set of visual and semantic changes, the user can think of each view of the information space as one of a series of frames; together, they are like a movie. Thus, the information space is a time-based as well as a spatial medium. As in other temporal media such as audio and video, it makes sense for the user to be able to seamlessly access the state of information space at different points in time. This thesis focuses on the interactive and computational semantics designed to facilitate the users in saving and restoring information space state, traversing the information space timeline and manipulating it. While designing these mechanisms, we have had these aims: to improve user's ability to design better information spaces with clear interactive semantics, to minimize cognitive load, and to facilitate the user's ability to see connections between information elements over time. While we focus on the user aspects, other work may deal more directly with cognitive and experiential aspects.

Section 2 gives a background of the related work in the areas of design of history systems, the uses of history mechanisms and a brief overview of non-linear video editing. Section 3 presents an overall summary of the ideas central to this thesis, ideas inspired from related work and a description of research contributions through this work.

Section 4 explores the interactive semantics of the information space authoring and generation over time. We have designed a set of new controls to enable linear time

traversal and non-linear history manipulation. These controls are based on tape-recorder metaphor and door-latch metaphor.

Section 5 describes the computational semantics developed to achieve the interactive affordances described above. To enable the users to visit any state of the information space in the timeline, we have developed a history mechanism. This history is the union of the program's generative actions, the user's design actions, and the user's interest expressions. As a result of the history, the state of the information space can be restored from any point in the timeline. The section examines the implementation details of the developed history mechanism including data-structures and algorithms used.

Section 6 describes the user study conducted to evaluate the designed history mechanism. It presents the findings of the study and the conclusions drawn from it. Section 7 summarizes the work done in the thesis, and future directions for time travel in generative information space are discussed.

2 BACKGROUND

In this section, we give an overview of the prior work done in the related areas of this research. The work done in this thesis draws from the existing history systems in text editors, browsers, hypertext tools, video and other specialized tools for collecting, organizing and manipulating information. This section also discusses previous analytical work done studying the usage and advantages of history systems.

2.1 History Systems

Text Editors such as Microsoft Word and others maintain the history of certain number of editing actions that have occurred until the current time. The user can go back a certain number of actions by hitting "Undo" multiple times or by selecting some number of "Undo" items at once. The editor actions are also maintained in the undo list and are undone normally like user actions. Some examples of editor actions are automatic spelling correction, capitalization of letters, indentation of paragraphs etc. Most editors however maintain only one branch. For instance, if the user undoes a certain number of actions and then performs some editing action, the subsequent actions which were just undone are lost and cannot be retrieved. This type of history system is called as stack-based history. Stack based models store the history actions linearly in the form of a stack. Thus, while they allow for history traversal in sequential order, they do not let the users maintain multiple branches or provide the ability to change the previous history actions. Any modification in the previous history action results in dropping of all the subsequent actions.

One of the most extensively used history systems is the access to history provided by traditional web browsers such as Internet Explorer and Netscape Navigator. Prior studies have shown that in these web browsers, the 'Back' button accounts for up to 42% of user actions [Catledge and Pitkow 1995; Tauscher and Greenberg 1997]. This shows how important the role of going back and forth while seeking for information on the internet is. Most web browsers implement stack-based history, but research has been done to study other ways of implementing history, such as temporal modeling [Cockburn et al. 2002], branching graphical history [Ayers and Stasko 1996], and zoomable graphical history [Hightower et al. 1998]. Temporal models store history actions based on time, as a result of which actions are not dropped when the user goes back and takes a new action. The graphical history system gives a graphical representation of the path traversed, along with all the branches. The work done in [Tauscher and Greenberg 1997] includes a detailed study of different history mechanisms used in the web browsers to lay an empirical foundation for their future design.

There are many examples in which the history system has been used to enhance the process of design. In a design process, the user might want to go back in time to see the state of the work. Klemmer's Designer's Outpost system provides a timeline annotated with thumbnails which the user can directly click and go to that particular time represented by the thumbnail [Klemmer et al. 2002]. The system supports branched history, but only the current branch is presented to the user as a linear history. This history is annotated with stubs, indicating the presence and position of other branches. The branches can be collapsed or expanded by the user.

The VKB system [Shipman et al. 2001] provides a slider, which the user can move on the timeline to restore the workspace back to that point in time [Shipman and Hseih 2000]. The history system in VKB supports only one branch. If the user goes back in time by using the history slider or by performing "Undo" multiple times and then starts manipulating the workspace, a new workspace is started from that point and the previous changes are lost; the history system does not reproduce those undone actions. The history in VKB also has a playback mode. Once the user goes back in time, s/he can hit the play button and choose an appropriate speed to play all the sequence of actions that have happened from that point until the current time in a movie-like fashion.

Rekimoto's Time-Machine Computing system describes a time-centric approach to organizing information on a computer's desktop [Rekimoto 1999]. It allows users to visit the different states of the desktop by maintaining a timeline. The time dimension accompanies the spatial desktop domain, in order to give an alternative perspective for managing and organizing information, in addition to the traditional folder hierarchies, which are common today. Some of the features related to history traversal provided by the system include:

- "Time-Travel" – ability to reconstruct the state of the desktop at any point in time.
- Time travel dial, back and forward buttons.
- Search back and forward.

The Editable Graphical Histories system by Kurlander and Feiner is a graphical history mechanism for a system called "Chimera" [Kurlander and Feiner 1990]. It

provides a visual representation of command histories of the “Chimera” editor using the comic-strip metaphor. An important fact to know about the system is that not all the changes are recorded in the panel containing the thumbnails. Instead it employs an inter-panel detail removal mechanism. The user can set the granularity level of how much coalescing of actions is performed. The visuals used in the panel are not exact snapshots of the editor screen. Instead an intra-panel content selection mechanism is used to choose the important objects in the scene. This allows for the users to easy visual recognition of the changes in the state of the system while revisiting states.

2.2 History Uses

Apart from the fact that history system helps the author himself/herself in the design process, it can also be used in other ways. The work discussed in [Reeves 1993] talks about how the history system is useful in a collaborative design process. Each author can see the difference in the state of the work since his/her last authoring session and quickly see what and how much has been changed by others. Sometimes, when there is an ambiguity regarding the motivation for the current state of the work, the author can play the history from the beginning and see how the information space has evolved and understand the current context.

Work done in [Lee 1992] includes a detailed study and analysis on the uses of information in a design history. It talks about different functions of history such as reuse, error-recovery, user-modeling, reminding and user-interface adaptation. The different existing history systems incorporate one or more of the above features as the use of their history system.

Greenberg and Witten did an extensive study to observe how often users repeat their actions on interactive systems. [Greenberg and Witten 1988]. They have chosen the example of UNIX command lines as a case study. The results obtained from the study are reformulated as empirically based general principles for designing history mechanisms, specifically for modern user interfaces.

The designers of VKB's history [Shipman and Hseih 2000] discuss the ability of the readers of the workspace authored by someone else to understand it better with the help of history. This is because the readers might not have an idea about how the author arrived at this final state. The history system helps in explaining the reader how the workspace has evolved from the beginning to the current state.

The art of Navigating through Hypertext by Jakob Nielsen observes how users become lost while exploring hypertext [Nielsen 1990]. It suggests mechanisms to avoid that, such as overview diagrams, interaction histories based on timestamp and footprints.

2.3 Non-linear Video Editing

The video timeline can also be seen as a history representing the set of frames in a movie. Editing video tape recorders and non-linear digital video editing systems typically include a jog-shuttle "wheel", which may be used to "scrub" the timeline, affording direct navigation to any point of time in the movie. Users engaged in editing or viewing use jog-shuttle when they are looking for a particular scene in the movie, or if they want to skip some parts of the movie.

Popular non-linear video editing systems such as Apple's Final Cut Pro allow non-linear editing operations such as copying and pasting clips, inserting between clips,

making overwrite edits and storyboard editing [Weynand 2004]. Different kinds of effects such as mixing, filtering, and cross-fading can be utilized to transform transitions across the clips.

3 RESEARCH CONTRIBUTION

In this section, we describe the ideas that are central to this thesis. We set out to give the user access to states of the information space, as it is developed over time. Since we began with no capability whatsoever for saving the state of spaces, the first step was to build a basic mechanism for save and restore of an information space. To do this in a general way that could support further development, we created the extensible ecologylab.xml translation framework. Another part of this stage was to enable publishing information spaces on the web, as *evolving metadocuments* that users could further generate and edit.

The experience of using combinFormation is as process-oriented as it is goal-oriented. Therefore, once we had developed the ability to save and open the spaces at a single moment, we became interested in the representation and manipulation of sessions of information space development over time. The first step in this development process was to design a linear history traversal mechanism, and extend the already existing tape-recorder metaphor by adding new preview-oriented controls for traversing the timeline. These include an implementation of the jog-shuttle mechanism that is popular in both analog and digital video editing systems. Once traversal of the space over time is possible, the user is prone to remembering or noticing things past, and desiring them in the present. We developed a new capability, based on the *door-latch metaphor*, for fixing elements across time. Because it changes history, this constitutes a form of non-linear time travel. We describe these three main ideas of evolving metadocuments, jog-shuttle slider with previews and the door-latch metaphor in the following sections. We

also compare our work with existing work, to indicate how our work was inspired by others, as well as to differentiate our contributions.

3.1 Evolving Metadocuments

Users of hypertext often need to collect references to significant places that they encounter while browsing. One can browse the web using combinFormation and compose references to different websites in the form of information samples in the generative space. These information spaces are documents that consist primarily of references to other documents, and elements within them [Bush 1945; Furuta et al. 1997]. In this sense, information spaces function as metadocuments. They consist of both, references by name, such as image elements and hyperlinks, which are represented by URLs and by value, in the form of textual quotations. Schraefel articulates the importance to metadocument authors of the connection between an information sample and its container [Schraefel et al. 2002]. When combinFormation collects information samples from the web, it maintains their source URLs, so it has the ability return to the sources of the quotations. Thus, each information sample can be thought of as an enhanced bookmark. The information space is a set of bookmarks, in the form of a composition. Before we developed the persistent storage mechanisms of this thesis, combinFormation information spaces lacked the capabilities of being able to be saved and re-opened for resuming the information space session; the information spaces were entirely process-oriented. The metadocuments existed only as transient states. To facilitate the sharing of these bookmarks we have developed an ability to save, publish and open these information spaces from the web. After saving the information space,

user-publishers can easily move them onto web servers. This is equivalent to what authors of regular HTML documents do when building a site. Now, once saved and published, they can serve both as static navigable metadocuments, and as the jumping off point from which the information space represented by the collected samples can continue to evolve. Thus, we call the ability to be able to publish the saved information spaces on the web as "Evolving Metadocuments" [Kerne et al. 2003]. The "readers" of these information spaces have access to the same expression and design tools as the original author, and the same generative capabilities. They can develop, save, and publish their own versions of the metadocument. One potential use is to enable publishers of large digital library collections to offer specific navigable evolving views of their libraries. Another is to enable users can derive and share their own collections and views. They may use these for themselves, and they may share these information space compositions of significant materials.

3.2 Showing Previews with History Slider

In the process of gathering information, the user does not always have a clear idea of the exact information needed by her/him. The user keeps deciding what information is significant while comparing materials from different sources [Ayers and Stasko 1995; Hightower et al. 1998; Kerne and Smith 2004]. combinFormation is a tool that provides a generative information space for collecting information resources, comparing them, and visualizing conceptual connections. Since the state of the generative information space keeps changing with time, and so does the cognitive state of the user, it may be useful to be able to go back and forth, seeing changes. We have

enabled the information space timeline to be traversed with a full set of tape recorder metaphor controls: *play*, *reverse*, *forward,* and *stop*. Additionally, to facilitate easy traversal and comparison across different states of the information space, we have developed a *jog-shuttle history slider,* with preview. This slider can be dragged on the timeline to visit different states of the information space. The slider helps the user by showing the snapshot of the information space at different points in time in the form of thumbnails. While moving the slider, a small preview window appears, overlaying the current information space. This preview window rapidly shows a series of snapshots of the information space state, with each matching the moment in time corresponding to the location of the slider. This provides a graphical aid to the users, and helps them in finding the right information while revisiting different states [Woodruff et al. 2001; Cockburn et al. 1999]. The details and a snapshot of the jog-shuttle slider with previews can be found in section 4.1.2.

Luesebrink introduced the idea of interactive time, to refer to time in which the reader is involved in meaningful exchange with a hypertext [Luesebrink 1998]. Shipman added constructive time [Shipman and Hseih 2000], addressing the author's writing process, which VKB presents to the user in the form of a timeline. We note that in the prior systems that provided a slider for traversing the history, dragging the slider across the timeline does not provide the user with visual cues about each state until s/he releases the slider there and waits for the system to rebuild that state. A cognitive burden of complex recall, or reading a textual history actions log, is imparted on the user. We expect that the visual preview provided while dragging the slider will increase the user's

sense of orientation while rapidly traversing linear history. Based on these previews, the user can decide whether s/he wants to go to that state, or traverse further. This can help to streamline the experience of interactive and constructive time in the generative hypermedia.

3.3 Novel Form of Non-linear History Manipulation

combinFormation is a mixed-initiative system [Horvitz 1999]. Both the user and program work in the same space to form a visual composition of information samples gathered from different sources. Operations include adding, removing, repositioning, resizing, and annotating samples. As a result of these manipulations, the information space changes. There might be some samples which were present at one point in time, but are no longer present at the current time. In order to move the information samples from one point in time to other, we have introduced the door-latch metaphor in combinFormation.

The *latch* tool can be used in conjunction with the history slider to fix the position of information samples from one state to the other, across time. Using the thumbnail previews which appear on moving the history slider, the user can look for and go back to the information space states that contain samples s/he wishes to utilize. The user can then latch the required information samples using the *latch* tool. When the information sample is latched, its position is fixed across time. That is, it becomes time invariant. Traversing across the timeline does not have any effect on a latched information sample. That is, the latch overrides all other operations on the sample that occurred in the history, as a result of which all the intermediate states are updated

automatically. In this way, the latch can be used to bring information samples from previous points in time to the current time or any other intermediate time in the information space timeline. Similarly the information samples from current time can be selected and incorporated to a previous state using the latch tool. We note that the latch overrides history replay operations that could have otherwise removed or modified the samples. Thus, we have provided a form of history manipulation using a novel interactive technique based on door-latch metaphor. The ability of moving information samples across time is one of the several possible forms of non-linear time traversal. Since the user's idea of needed information evolves over time, s/he is likely to realize later about what is important and what is the needed information. Some of the needed information samples might have been existed in the past. That latch tool along with the information space history facilitates finding and retrieving information samples from any point in the timeline.

4 INTERACTIVE SEMANTICS

This section describes the interactive semantics of linear time traversal and non-linear history manipulation based on tape-recorder and door-latch metaphors.

Linear time traversal enables users to visit any point in the information space timeline. This gives users the ability to experience the whole process by which the information space has evolved, rather than just the current state. The history timeline may be useful for the authors of the information space for reviewing previous states, going back and starting over. A reader of the information space will get more information from a space if s/he has access to the complete history rather than just the final state of information space [Shipman and Hseih 2000].

After implementing the linear history traversal features, it seemed natural to explore the non-linear history manipulation features to enhance the experience of the user working with the combinFormation information space. Ability to only visit different states would not help too much in designing better information spaces. Thus, we have explored a few forms of non-linear history manipulation and implemented one such feature in the current work.

4.1 Linear Time Traversal

To facilitate linear time traversal in combinFormation information spaces, we have extended the existing interactive controls. Previously, the software provided the *play* and *pause* controls along with the *rate slider* [Kerne and Sundaram 2003]. We have now included additional controls of *forward*, *reverse*, *reset*, *jog-shuttle slider* and a *time*

counter. These new controls complete the tape recorder metaphor that combinFormation was already using and provide the users with the ability to go back and forth in the timeline.

4.1.1 Tape Recorder Metaphor

The foundation of the controls provided to operate the combinFormation timeline is the tape recorder metaphor as shown in Figure 2. Each component of the tap-recorder metaphor is described below:

FIG. 2. Tape Recorder Metaphor Controls in combinFormation. [Kerne and Interface Ecology Lab 2004]

Record

Record works like the normal record button in an audio-recorder. It is the same as the play button in previous versions of combinFormation. Record means move forward in generating the information space. Record is activated automatically at the start of a typical combinFormation session, when the program brings in information samples based on seeding. When the user is at end of the timeline, information space generation is appended to end of the history that s/he has created so far. If s/he has moved back in time, using the reverse or jog-shuttle controls, record overwrites the part of the information space history timeline that existed after that time, as it generates new

forms of the space. Notice that, just like in most audio-recorders, hitting the record button also pushes the play button automatically; they go together.

Forward Play

If the user is at the end of the timeline, hitting play will activate record and continue generation of the space. In other words, it automatically pushes the record button and starts building a new information space. If s/he has moved back in time, and hits play button, it replays the sequence of agent and user events that had been performed from that point in the timeline. It affords backwards linear traversal through the history. It can be used to see how the information space has evolved since some previous point in time.

Reverse Play

Reverse plays backwards through the sequence of information space composition events that happened prior to the current point in the timeline. It affords backwards linear traversal through the history. This can be seen as a way of doing "undo" multiple times. Both users' actions, and those of the agent, are included.

Note that playing forward or reverse is different from restoring the information space state using the jog-shuttle slider. This is because, while playing forward or reverse, the space is regenerated one operation at a time with the same time lag between the operations when the operations were actually happening in "record mode". Whereas in the case of jog-shuttle, the complete information space from the selected point is restored at once (See section 4.1.2)

Pause

Pause temporarily stops generation of the information space, or other forms of temporal traversal. The user can use this when s/he wants complete control, rather than sharing control with the agent. Because the information space can be paused anytime and resumed, there are two notions of time possible on the information space timeline. One notion is the "total time" which represents the actual clock time. This includes the time when the information space was being recorded as well as the time when it was paused. The other notion of time is the "elapsed time", which is the time for which the information space was running in the generative mode, i.e. with information space playing in the "record" mode.

Rate Slider

The rate slider is provided to control how fast the generation of the information space (in record mode), or traversal of history through forward or reverse playback proceeds. This is a way for the user to affect how much control s/he has, and how much control the program will take, in composing the information space over time. This ability to control the rate introduces non-linearity in the combinFormation timeline. The rate at which the space was recorded can be different at different times. There can be some time when nothing is recorded (when the space is paused). Due to this difference in speeds, the information space contents are not directly proportional to the time for which it is being recorded. This concept of "vari-speed" introduces a few issues in the way timeline works and is discussed in the jog-shuttle slider section 4.1.2.

Reset

combinFormation provides hot and cool areas within the information space. The hot space is the central area on the screen, where both, the agent and the user can perform operations. The agent can add or remove information samples from this area and the user can manipulate the samples on the same space with the help of different design and interest expression tools. The cool space is the grey border around the hot space on screen. The hot and cool areas can be seen in Figure 3, where the central white area is the hot space while the grey area surrounding it is the cool space. The cool space is exclusive to the user; the agent does not perform any action on this part of screen. The proportion of screen size of the hot and cool areas can be changed via a preference that the user can set when starting up combinFormation. The Reset button performs a visual clear of the hot space. Elements in the cool space are not affected.

4.1.2 Jog-Shuttle History Slider

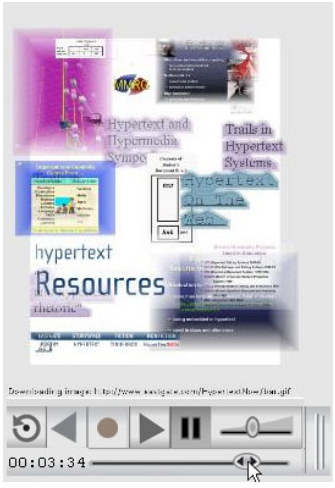

FIG. 3. Preview with the Jog-Shuttle History Slider. The image shows the snapshot of the preview window displayed by combinFormation when the user goes back in the timeline in an information space created by giving the search term “hypertext”. [Kerne and Interface Ecology Lab 2004]

We provide a Jog-Shuttle slider using which the user can use to continuously traverse the timeline combinFormation browsing session. The user can go back and forth in the timeline by simply dragging the knob over the slider. When s/he moves the knob on the slider, a small preview as shown in Figure 3 is periodically displayed just above the slider which shows the state of the information space from that point in time. When s/he releases the knob, the information space is restored to the state shown in the preview. Complete restoration may take time, as images may need to be retrieved again.

We note that the visual look of the combinFormation interface provided for time travel features is different from other systems such as Chimera and Designer's Outpost, which provide similar features for traversing the history. In combinFormation, initially the user can see the information space and the different control panels provided consisting of composition and interest-expression tools and the other one consisting of tape-recorder metaphor tools. The preview images of the information space are shown when the user drags the jog-shuttle history slider. These previews correspond to the position of the knob on the slider. As opposed to Chimera and Designer's Outpost [Kurlander and Feiner 1990; Klemmer et al. 2002], we do not show the series of thumbnails on the panel. This helps us in avoiding the problems of dealing with too many thumbnails and the problem of optimizing the screen real estate for presentation of these thumbnail previews since in our system they get shown only "on-demand". On the other hand, since the user cannot see all the preview thumbnails on a panel, they cannot make a side by side comparison of all the intermediate states of the information space at

once, which is possible in other systems. We have discussed a possible way of showing all the keyframes previews at once, in the Future Work section in Section 7.

Due to the concept of "vari-speed" due to which the information space could be recorded at different speeds at different time, the jog-shuttle slider is non-linear. That is the density of operations throughout the slider length is not uniform. There might be some regions of the slider where a lot of operations might have occurred; while there might be others where the number of operations occurred could be very small. Thus, the jog-shuttle slider can be used to represent time or it can be used to represent the mixed-initiative operations of the information space. By mixed-initiative operations, we mean the entire set of the user's interactive modifications to the information space, as well as the generative actions the program performs on the information space. These mixed-initiative operations are stored by the program in the form of a series of keyframes and operations and the details of this representation are described in the section 5. For the purpose of understanding of issues in this section, a keyframe can be thought of as a bundle of 25 mixed-initiative operations. Detailed explanation on how the keyframe interval is determined is presented in section 5.5. The two representations are discussed below:

Slider representing time: If the slider represents time, then on moving the knob, the program should calculate the time corresponding to the position of the knob and restore the information space from that point in time. The start of the slider will represent the time when the information space was started (time t = 0) and the end of the slider will represent the current time (time t = *t*). Now if the user moves the knob half

way back from the end position, the program should retrieve the keyframe emitted at the elapsed time *t/2* and display that on screen. Note that if the total number of keyframes in the space is *k* at the current time, the keyframe emitted at time may not be the $k/2^{nd}$ keyframe. This is because the ability to record the information space at different speeds at different times makes the information space timeline non-linear. As a result, the position of the knob on the jog-shuttle slider is not directly proportional to the elapsed time of the information space. Further, even within the representation of slider representing time, there can be a question as to whether the time represented should be the total time or the elapsed time because of the difference between the two notions of time explained in section 4.1.1.

Slider representing mixed-initiative operations: If the slider represents mixed-initiative operations, then on moving the knob, the program should take the information space back/forward in terms of the mixed-initiative operations performed on the information space. In other words, in this representation, the start of the slider represents the 0^{th} keyframe and the end of the slider represents the last emitted keyframe (*k*). If the user moves the knob half way back from the end position, the program should get the keyframe number *k/2* and display that. Note here again that, keyframe number *k/2* may not have been emitted at elapsed time *t/2* because of the non-linearity in the information space timeline discussed above.

It can be a subject of debate as to which representation is better. A user study could be performed to evaluate the usefulness of either of the two representations. In the perfect world, we would have supported both the representation with the users having

the ability to switch back and forth between the two representations. In the current implementation, the slider represents the mixed-initiative operations. The main reason of supporting that is efficiency and simplicity. According to our architecture, to retrieve the state at any point in time, we need to restore the nearest keyframe corresponding to the location of the knob on the slider. To do this, we calculate the position of the knob on the slider on a scale of 0 through 1 and multiply this fraction with the total number of keyframes at this point. This gives the position of the keyframe that corresponds to this knob location. This way, finding out the required keyframe is a constant time operation. We then retrieve this keyframe and restore it. If we were to support the elapsed time representation of the slider, we would have needed to search for the keyframe, which was emitted at elapsed time *t/2*, and display that. This would be a linear time operation. Further, we thought that the slider representing the mixed-initiative operations would be more intuitive to the user since moving the slider by constant amount will change the information space by similar amount in terms of content. This is because the distance on slider is directly proportional to the number of mixed-initiative operations.

Time Counter

The jog-shuttle slider also has a counter next to it, which displays a time on it. Questions arise as to which time should this counter display. Should it display the total time or the elapsed time? Given the fact that we decided to choose the slider representing mixed-initiative operations, it makes more sense for the counter to display the elapsed time rather than the total time. This is because elapsed time is updated only when the system is running in generative mode while the total time keeps updating even when the

system is not taking any generative operations. This time counter is updated when the information space is running in record, play back or forward mode or when the user is traveling back and forth in the timeline using the jog-shuttle slider. The presence of this counter gives a sense of time-orientation to the users of the information space.

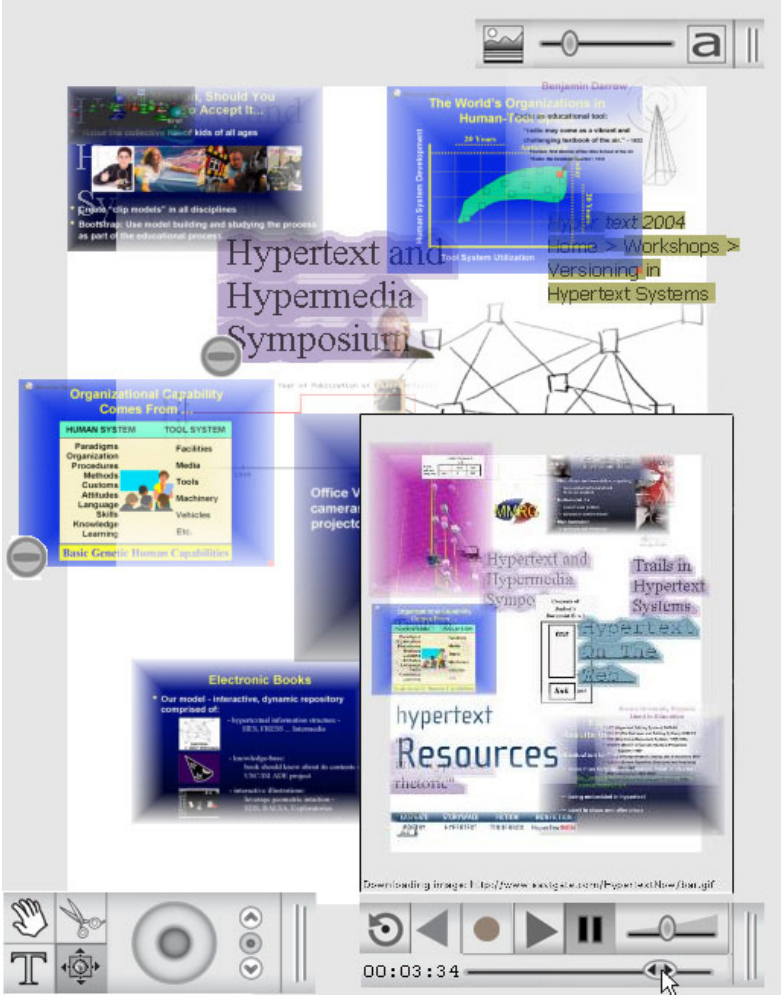

FIG. 4. combinFormation Information Space with Tape Recorder Controls. The user is in the midst of dragging the Jog shuttle. A preview window is shown overlaying the current information space. The image was created using the combinFormation software by giving the search term "hypertext" [Kerne and Interface Ecology Lab 2004]

A combinFormation information space is shown in Figure 4 with the design and expression toolbars and the tape recorder controls. The figure also shows a preview that appears on dragging the jog-shuttle slider.

4.2 Non-linear History Manipulation

In order to understand the semantics of non-linear history in combinFormation, we begin by considering and describing hypothetical scenarios in which elements and actions during history traversal change the flow of time. Through these discussions, we develop an understanding of different forms of non-linear history possible in our context. From this set of possibilities, we will proceed to present the currently implemented work, as well as ideas about how it can be further extended.

4.2.1 Meaning of Going Back in Time in a combinFormation Information Space

When the user goes back in time, the program could take the following actions to restore the state of the information space from the previous point in time.

- Restore the visual state of the information space. The system must go back to the exact visual state as the previous time. This means that the same samples must be present on screen with the same visual attributes such as location, size, color, transparency and blending effects.
- The user interest model must be restored. Over time the program forms a model of the user, based on interactive expressions of interest. The expressive mechanisms include the positive, negative and neutral expression states on each of the information samples and terms, image-mix ratio and the new terms created

by the user, using the text-tool. Going back in time means the system should forget about the interest/disinterest expressed on each of the information samples and terms, and thus get back to exactly the same state as it was at the previous time.

- Restore the media collection as it was at the previous point in time. As a result of interest/disinterest expressed by the user, the system forms a model and crawls the web to bring information from the internet that might be interesting to the users. In the course of procedural information collecting [Kerne and Sundaram 2003], some of the information samples that are gathered by the system are displayed on screen. Others remain in internal collections, only. In the normal operation of the procedural visual composition component, once an information sample has been chosen for display, it is prohibited from being displayed again. As part of restoring the media collection, the program must restore state so those samples can be displayed again. Going back in time should undo all the crawling and display choices that have occurred between the previous time and the current time, which ensures that the system has exactly the same set of available media collection at the time which user went back to.

4.2.2 Notions of Non-linear History in a combinFormation Information Space

Another point of discussion is: once the user goes back in time, what situations that interleave history traversal with interaction are possible? What courses of action could be available to users?

1. The user goes back in time. At some point s/he changes the interest state of some samples by using the interest expression tools, and starts recording again. In this case, it is necessary for the system to restore the complete user interest model, as well as the media collection (from which the information samples to be displayed are chosen) from the previous point. This would ensure that the system is exactly in the same state; it can start building the information space as if nothing had happened after that. Since the interest state on some information samples has been changed, it is likely that the new information space produced will be different from the previous one. Thus, this is a start of a new branch.
2. The user might be interested in "editing" the history. By this we mean that s/he might want to remove, add, resize or change the position of information samples, after going back in time. S/he would perform edit or interest expression operations, without running combinFormation in "record" mode.
3. When the user goes back to the past, s/he may see a sample that disappeared, but which s/he now wants in the composition. Instead of starting a new branch, the system needs to edit the existing one, by bringing samples from the past into the present. In this case, it is not necessary to restore the model and media collection, as the user just needs to move the samples across time.

Because they change the history of the information space, each of these scenarios can be construed as forms of non-linear history. These forms of non-linear history can enhance generative information space experience, by giving the user new perspective and mechanisms for manipulating the composition across time.

Scenario 1 would be computationally expensive to implement, because it requires precise journaling of all the behind-the-scenes actions of the combinFormation agent, in addition to the visual actions of the agent and the user. The agent maintains collections that are orders of magnitude larger than the set of samples that gets visualized. The temporal density of behind the scenes operations on the collections is also much greater than the rate of visible changes to the information space. All this work is out of the scope of this thesis. In the current implementation, when the user goes back in time and starts the combinFormation in "record" mode, a new information space is started from that point and the subsequent history is deleted. We do not support maintaining multiple branches of the information space session.

Although Scenario 2 has not been implemented yet, the underlying infrastructure used to represent the history does provide a basis for this, as future work. In the current implementation, we only support Scenario 3, using the door latch metaphor. The latch metaphor semantics and a use case are described below:

4.2.3 The Latch Metaphor

We introduce interactive affordances based on a *door-latch* metaphor. We provide a latch tool that locks an information sample at a fixed position *across time*. This latch tool is one of the several tools provided in the form of *local tools*. Local tools are the tools that appear local to each information sample. So the effect of any of these local tools are limited to the information sample to which it may be associated with. Other local tools include *search tool* to initiate search related to the information sample,

changing the size and font of text information samples and enabling or disabling blending of the information sample.

The latch tool appears when the user rolls over the mouse on an information sample. Initially an open latch is seen as a tool appearing next to the sample. When the user clicks on the latch, the closed latch is applied on top right corner of the information sample. These states of the latch tool are shown in Figure 5. The latch overrides history replay operations that would have removed or modified the sample. It can be used to bring information samples from the past, back to the future, or vice-versa. It also has the effect of extending the cool space to single elements anywhere within the information space.

FIG. 5. Three States of the Latch Tool. First the latch tool appears on mouse rollover. Second, it can be applied by clicking on the tool. Third, closed latch on a latched sample. The image is one of the images that combinFormation software retrieved from the Web on giving the search term "movies". [Kerne and Interface Ecology Lab 2004]

For example, let us say that the user is working on an information space. As s/he is working on composing an information space, s/he recalls a related information sample from some previous point in time. That sample is no longer present on screen, but the user wants to use it in the composition. In this case, the user can go back using the jog-shuttle and see different states of the information space in the keyframe preview. Once the user finds the sample s/he is looking for, s/he can release the jog-shuttle knob and restore the information space from that time. Then, s/he can latch the required sample(s) using the latch tool. S/he can then proceed to jog forward (back to the future), to what had been the current time before all this jogging started. Since the sample was locked, it will remain in the current information space along with the other samples. The latch tool can also be used to lock an information sample at a fixed position during the normal generation of the space.

5 COMPUTATIONAL SEMANTICS

To enable the interactive mechanisms described in the previous section, development of under the hood data-structures and algorithms, to be able to represent the history of the information space was required. We needed to develop extensible and easy-to-maintain computational representations, so that new features can be incorporated easily with the continuing development of the overall combinFormation software. These representations serve as the foundation on top of which, the front-end interaction mechanisms are developed. The semantics of these computational representations are described in this section.

To store and retrieve information space histories, we need to be able to save the data structures representing the information samples persistently. As the structure of the model changes, the data needed to be saved for information space storage and retrieval also change. We were also concerned about software engineering issues, because combinFormation and the structures it needs to store to represent information space state over time are extensive, and in an ongoing state of flux. That is, we continue to develop the software, and with it, the underlying representations which make up the semantics of information space state.

XML supports extensible hierarchical structures in the form of the Document Object Model (DOM) [DOM]. Both XML and DOM are open standards and parsers are widely available [XML]. Due to these reasons, we have chosen XML to represent the program data structures persistently in the file system.

Java supports Reflection, which enables Java programs to read and manipulate the structure of Java objects [Gosling 1996] at runtime. During translation from Java to XML, it allows us to peek into the data structures at runtime and determine the names of the fields as well as values. While converting from XML to Java, the Reflection API allows us to dynamically create Java objects at runtime, just using the class names. It also allows the program to invoke their methods. This is important, because usually programmers must know the names of objects, fields, and methods while they are programming, in order to use them.

Using these technologies, we have created an extensible framework called ecologylab.xml, which makes it easy for developers to incorporate the changing nature of structures and automate the process of translating Java to XML and back. The flowchart of the conversion process is shown in Figure 6 and the process of conversion is described in the following section.

5.1 ecologylab.xml: Java-XML-Java Translation Framework

ecologylab.xml uses recursive descent algorithms that automatically convert Java objects to XML, and back. The framework imposes a one-to-one mapping between the Java class-structure and the XML hierarchy. It also automatically utilizes and enforces a rule-based mapping between names of Java classes and instance-variables, on the one hand, and XML element names and attributes, on the other. Because of this automatic mapping, the programmer does not need to take care of creating names for XML elements corresponding to Java data structures. This minimizes the chances of human errors such as typographical mistakes, and clashes or inconsistencies in naming schemes.

As a result of this, data structures are written back and populated correctly. The generated names are methodically consistent and intuitive.

For translating Java objects to XML, the framework uses a top-down recursive descent algorithm, which walks the Java data structures and emits an XML string corresponding to the encountered Java structures. For the reverse translation from XML to Java, the bottom-up recursive descent algorithm walks the XML DOM tree, and builds Java data structures as it translates from XML. We describe these algorithms in the following section.

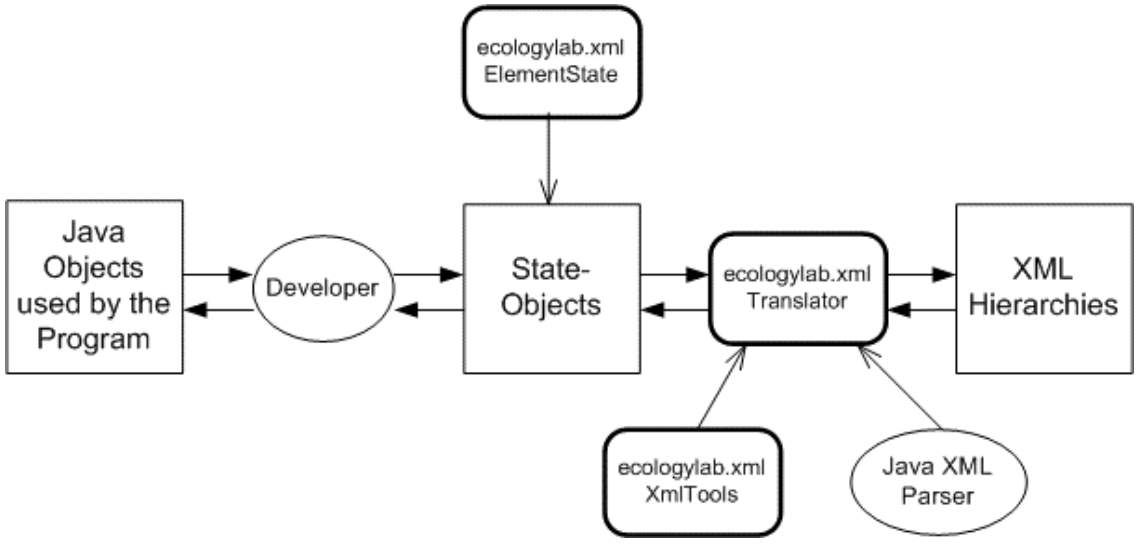

FIG. 6. ecologylab.xml Java-XML-Java Translation Framework.

5.1.1 Translation of Java Objects to XML

The generation of XML is a two-step process. The first step is to convert the Java objects to be translated to XML to an intermediate representation called "State-Objects". The next step is to convert the State-Objects to XML.

Step 1: Formation of Intermediate Objects called State-Objects

The developer using the framework must create intermediate objects called State-Objects. State-Objects are intermediate objects, which must have the same hierarchy as the functional data structures that need to be stored persistently. State-Objects are different from the functional Java objects in that, they are not actually used by the program in its normal course of execution. The developer must create State-Objects only for the objects that need to be saved persistently and must include only the data that those objects need to save. These are used only during the translation process. Thus, the definition of State-Objects serves as a specification for the XML; that is, the XML structure is derived from the structure of the State-Objects. Each of these objects must be derived from the `ElementState` base-class provided by the ecologylab.xml framework. In order for the framework to work properly, the developer must follow the following conventions while creating the State-Objects.

- All State-Objects must have an empty constructor. This is used by the framework while translating the XML to Java objects.
- All fields that need to be stored persistently (as XML) must be declared with the modifier `public`. This is because the Reflection API cannot find the values of non-public fields at runtime.

- Within each State-Object, all instance variables of primitive types must be declared first. They are followed by reference-type fields, which denote nested objects/elements. For the purpose of this translation process, along with the standard primitive types defined by Java (`int, float, double, char, byte`), we also consider the types `String`, `URL`, `Color, and Date` as primitive types. This is required because, by convention our framework emits all the primitive types as the fields of its parent object and it emits sub-elements for reference type fields. Since attributes must be stored in the XML before the start of any sub-element, all primitive type fields need to be declared before any reference type fields.
- The order in which child reference-type objects are declared inside the State-Object is significant, because the ecologylab.xml translation framework uses a bottom-up recursive algorithm to create objects. The functional Java objects are built in the same order when translating from XML back to Java objects. If one object depends on another for its construction, it should be specified after its dependent object as a field in the parent object.
- Corresponding to each primitive field `foo` to be emitted as XML, within the State-Object, there must be a method with signature: `public void setFoo(String value)`, which parses the string value and appropriately sets the value of the `foo` depending on its data-type. This method is called by the framework to set the values inside the object when converting from XML to

Java. For example, if `foo` is of type `int`, the method might be defined as follows:

```
public void setFoo(String value){
        foo = Integer.parseInt(value);
}
```

- Each State-Object must also have a method called `buildRealObject(ElementState parentElementState)`, which has the mechanism for building the functional Java object that the State-Object corresponds to, using the fields populated in the State-Object by the XML-Java translator. If this method is not present, the translator will not be able to build the functional Java object from the State-Object.
- If one State-Object has a collection of another type of State-Objects inside it,
  - It must have a method called `getCollection()`, which returns the `iterator` to that collection, so that the translator can iterate over it and generate XML for each item in the collection.
  - It must override the `addElement(ElementState State)`, method which adds the objects appropriately, depending on the type of the collection. This is because, the method of adding an object to a collection type object is different for different types of collections. For example, in `Vector` it is `add()`, in `Hashtable` it is `put()`.

This rule, helps our framework to handle any type of collection transparently.

Step 2: Translation of State-Objects to XML

The class `ElementState` provided in the framework is the base-class for all State-Objects. In `Elementstate` class there is a method called `emitXml()`, which recursively descends through the list of fields in the State-Object in a top-down manner, and emits an XML attribute or element for each field. `Class` and `Field` name translation rules transform Java class name descriptors, such as `MediaElementState`, into XML element descriptors, such as `media_element`. Whenever the `emitXml()` translator encounters a primitive data-type, the framework generates an XML attribute name-value pair in that element. When it encounters a reference data-type which is derived from State-Object, it starts building a nested XML element. `emitXml()` descends into each object recursively, generating XML attributes for primitive data-types. For each field corresponding to a reference data-type it generates a child-element in the XML. If the field is of a `Collection` type, (e.g. `Vector`, `Hashtable`), `emitXml()` iterates over the collection and emits a nested XML element for each of the object in the collection. The following figure illustrates an example of a Java `State-Object` and the corresponding XML generated by the translator.

```
public class Record
extends ElementState{
   public int idNumber;
   public String name;
   public Address address
   public ContactList contactList;
}
public class Address
extends ElementState{
   public Street street;
   public City city;
}
public class Street
extends ElementState{
   public String name;
}
public class City
extends ElementState{
   public String name;
}
public class Friend
extends ElementState{
   public String name;
}
public class ContactList
extends ElementState{
   public Vector list;
}
```

```
<?xml version="1.0"?>
<record idNumber=232 name=”abc”>
    <address>
       <street name=”j.f.k blvd”>
       <city name=”xmlVille”>
    </address>

    <contact_list>
       <friend name=”java”>
       <friend name=”c++”>
       <friend name=”perl”>
    </contact_list>
</record>
```

FIG. 7. A Java Class and Its XML Representation.

In Figure 7, we see a Java `class` on the left hand side. The `class Record` contains two instance variables of primitive types (extended primitive type defined earlier, which includes `String`) `int` and a `String`. In the corresponding XML on the right hand side, we see that an XML attribute name value pair is generated for these primitive types. `Address` is a reference type data structure, which internally contains `Street` and `City` classes. An XML sub-element is generated for the instance variable of type `Address`. Similarly for the classes `Street` and `City`, an XML sub-element in is generated. `contactList` is a `Vector` containing a collection of classes of type

`Friend`. Also noteworthy is how the XML element names are generated automatically. For example, `contactList` is translated to `contact_list` automatically by the translator.

Sometimes, when large data needs to be emitted in the form of XML, and long class names are used in the program, the XML representation can quickly become verbose and bulky. To keep the size of the emitted XML small, we have provided an option, by which the user can supply a compression table to the framework. This compression table is a set of name value pairs, where name is the name of the class in the program that is being translated and value is the abbreviated code for the class that will be used by the translator as the name of the corresponding XML element, when converting to XML. For example, for a class name like `MediaElement`, the translator will generate `media_element` as the name of the element in the XML. If there are a large number of such lengthy element names, the user can provide an abbreviation such as `m_e` for `MediaElement`. This way, the developer can control the size of the XML file, if it is growing too big rapidly.

Another way of keeping the size of the emitted XML small is by not emitting the default values of the fields. The default value of any primitive type field can be set by the developer using a method provided in the framework. When the translator encounters a field that has the default value as supplied by the developer, the field is not emitted in the XML. While building the Java objects back from XML, the values of the fields not populated by XML translator are automatically set to their default values.

5.1.2 Building Java Objects from XML

As mentioned before, the framework also provides facilities to automatically generate Java objects from their XML representations. This process is the reverse of generating XML from Java and is explained in the following two steps:

Step 1: Formation of State-Objects from the XML

The framework first builds State-Objects corresponding to the XML elements that were previously emitted. To achieve this, the algorithm reads the XML in a DOM and walks through the DOM tree recursively, in a bottom-up manner, to build the State-Objects for each XML element. The field values of primitive types in the State-Objects are populated by the attributes in the XML. The reference data-types are first constructed using the empty constructor present in the State-Object, and then filled out from the child-element structures in the XML.

Step 2: Building Functional Data Structures from State-Objects

Each State-Object must override a method called `buildRealObject(ElementState parentElementState)`. This method is responsible for building functional data structures from the data stored in the State-Object. Thus once the State-Objects are formed, the framework gives the control to `buildRealObject(ElementState parentElementState)`. This process cannot be fully automated as the framework may not have any idea about the working of the software it is used in conjunction with. `buildRealObject(ElementState parentElementState)` for a `Collection` type of State-Object iterates through

the collection and builds the functional object for each of the element in the collection. This is the final backend of the parsing process. It is prone to requiring the most hand-coding on the part of the developer, because here the program must generate and incorporate objects using its own internal semantics. This is similar to the backend of a compiler built with Lex and YACC [Levine et al. 1992]. It corresponds to the actual YACC production rules.

5.1.3 Comparison with Other Similar Frameworks

There are several other existing frameworks such as JOX, KBML, BeanML, and JAXB that provide functionalities similar to the ecologylab.xml framework. These are the ones that are most related ones in terms of the functionalities offered. We give a brief description of each of these frameworks and highlight the similarities and differences with our framework.

JOX: Java Object Serialization Using XML

JOX provides similar functionalities as ecologylab.xml. It allows translation of Java objects to XML form and reverse translation of XML to Java objects [JOX]. Specifically, it converts Java Bean Objects to XML and back to Java Bean. To read the fields inside Java objects it uses "introspection" using facilities provided by the Bean as opposed to the Reflection API that we use. This framework is slightly more flexible in terms of mapping between the field name and corresponding XML tag names, in that they support "-" as a delimiter, in addition to "_". For example, there may be a `Person` Bean that has `firstName` and `lastName` properties. You could read in an XML file

containing `<first-name>` and `<last-name>` fields. You could also read the XML file into a `Customer` bean that has `First_Name` and `Last_Name` fields. Additionally, a DTD can also be defined to specify a mapping between the Java field names and XML tag names. The framework is similar to ecologylab.xml in terms of handling nested objects. One limitation of JOX is that it cannot handle `Hashtables` in the object being translated, while ecologylab.xml framework can handle abstract collection type objects in one type, and define a set of constraints in the `State-Objects` to work with such collections in a generic way. Thus, ecologylab.xml framework can handle any collection type object that implements the `Collection` interface defined by Java.

KBML: Koala Bean Markup Language

KBML also converts Java Beans to XML and back [KBML]. This framework, instead of using the field names for creating tag name in the XML uses "name" and "value" explicitly in each field, for example:

```
<property name="email">
<value id="KBML_9">Philippe.Kaplan@sophia.inria.fr</value>
</property>
```

We note that because of this naming scheme in the KBML framework, the generated XML can become quite verbose. A significant point to note about both JOX and KBML is that they use the Java Bean specification to enforce rules such as defining set methods on objects being converted to XML, whereas we define our own set of constraints by the State-Object specifications. Both methods have their own advantages

and disadvantages. Using Bean specifications is good as it is an industry standard and so it will have widespread acceptability, while defining our own specifications gives flexibility in the design and easy extensibility.

BeanML: Bean Markup Language

There have been some industry level efforts by corporations such as IBM and Sun to provide data binding between Java objects and XML. IBM has given specifications for Bean Markup Language [BeanML], but currently it supports only translation from XML to Java, the reverse is not supported.

JAXB: Java Architecture for XML Binding

Sun has defined JAXB [JAXB], which gives wider functionalities such as validation, but requires a DTD or XML Schema to generate classes that handle the XML data. It also needs the classes being translated to implement specific methods such as `marshal` and `unmarshal`, to take care of serialization and deserialization. Our solution do not require users to explicitly take care of serialization and deserialization. However, we do need the developer to comply with the naming and ordering conventions of the fields inside a Java object and implementation of trivial set methods (described in the previous sections) for setting the values of the fields. We also observe that while the JAXB specifications are defined at a bigger scale; they require extra skills and efforts from programmers. Unless such functionalities are required, they can tend to become too tedious and heavyweight.

Size Comparision with Other Frameworks

We note that ecologylab.xml offers a significant size reduction compared to the above mentioned frameworks. The size of the ecologylab.xml jar file is 16.5KB. This is smaller than JOX, which is 39KB, KBML, which is 37KB, BeanML, which is 70KB and JAXB, which is 280KB.

5.2 Files in the ecologylab.xml Package

We have completely isolated the functionalities of the XML translation framework from the rest of the combinFormation code. A Java package named `ecologylab.xml` has been created. This package and derived classes are used in combinFormation for several functions: saving and opening information spaces, creating logs of user activity for studies, and for the history mechanisms which are the focus of this thesis. The ecologylab.xml package may also be useful to others who need to perform Java-XML-Java translation of complex nested data structures. Thus, we intend to open source this package under an appropriate license so that it can be obtained, used, and extended by others. The ecologylab.xml package contains the following Java classes:

- `ElementState.java`: This class is the heart of the translation framework. All classes which need to be translated to XML and back must extend this class. It includes the actual methods, that emit the XML from Java objects (`emitXML()`) and build the Java classes back from XML (`buildStateObject(ElementState parentElementState)`)

- `XmlTools.java`: This class contains methods, which are used during the translation of Java objects to XML and back. The compression table mentioned earlier, which is used to compress the produced XML files, can be supplied to a method called `setCompressionTable` in this class.
- `IO.java`: This class contains methods for doing all the I/O required. It is used internally by the other classes. It has facilities for printing debug messages and the File I/O facilities required for XML reading and writing.
- `XMLTranslationConfig.java`: Developer can use this class to store the mapping of the class names with their package names. As a result of this, the package names need not be stored with the XML. This helps in keeping the size of the XML small. If the developer does not maintain the package information in this class, the program prepends the package name of each class to the corresponding element in the XML, resulting in bigger XML files.
- `XmlTranslationException.java`: There are a set of rules one has to follow while using this framework for translation from Java to XML and back. This `Exception class` enforces those rules on the developer, and provides informative error messages to guide the developer in case s/he has made a mistake. So, if the user fails to follow any of the rules, this `Exception` is thrown with appropriate error message encapsulated in it. This exception is thrown in the following conditions:
    - If the user is translating a class which has a `Collection` type object in it (e.g. `Vector, Hashtable`) and the class does not override the

methods `getCollection()` and `addElement()` inherited from `ElementState`.

- If the user is translating a class, which has an object that is neither a primitive type, nor one derived from the class `ElementState`.
- The class to be translated does not provide an empty `constructor`. In this case the exception is thrown only when the user is building Java classes back from XML.
- The class to be translated does not provide the required `set` method for setting the value of a primitive type variable which is being translated. In this case the exception is thrown only when the user is building Java classes from XML.

5.3 Saving a combinFormation Information Space State

ecologylab.xml translation framework described in the previous section is used to save the state of the information space. For saving any state of the information space, so that it can be opened later from the same point, the program needs store the following information:

- On-screen information samples along with their visual attributes such as location, size, and color (for text chunks).
- References to the containers (source web documents) from which the information samples were extracted and the hyperlinks they may be pointing to.
- The user interest model associated with each of the information samples and the containers.

- Terms and interest levels stored by the software, which constitute the information retrieval portion of its user interest model.

When combinFormation software is running, it maintains a linked list of information samples that are currently displayed on screen. Each object in the linked list contains all the information about that sample. To save all the above mentioned data, the program traverses this linked list and stores all the necessary information of each of the element in the list. Appropriate State-Objects are created for each visual element; the XML translation framework takes care of generating the XML. The program also saves an image file (JPEG format) representing the visual snapshot of the information space. The XML file is useful, if the user is interested in opening this saved information space at a later point. The JPEG file is useful for the users, who are interested in saving the current screenshot of the information space.

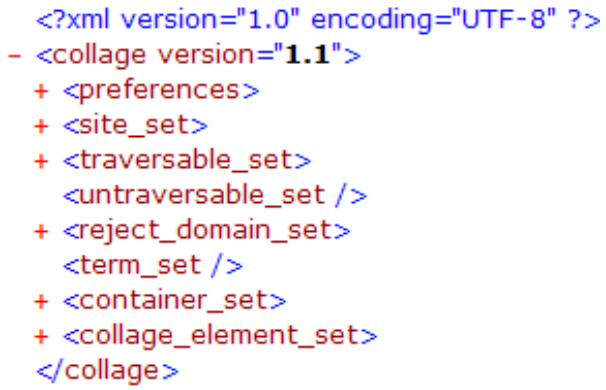

FIG. 8. XML Representation of an Information Space without History.

Figure 8 shows the structure of XML file for a saved information space without history. The information contained in the file is explained below:

- `preferences`: This stores the resolution of the screen and the size of the information space.
- `site_set`, `traversable_set`, `untraversable_set`, `reject_domain_set`: All these sub-elements contain information regarding which sites and domains are allowed for the combinFormation parser to traverse and which are not and thus direct the web crawler accordingly.
- `term_set`: This is the set of information retrieval terms associated with this information space, which the user has expressed interest in.
- `container_set`: This is the set of container source documents (HTML pages, PDF pages) from where the information samples in this information space were actually extracted from.
- `collage_element_set`: This is the set of information samples that were actually present on screen. Each information sample is represented in XML as depicted in Figure 9.

```
- <collage_element id="http://news.bbc.co.ukAristide departs for South Africa"
    containerURL="http://news.bbc.co.uk" historyNum="6">
  - <media_element>
    - <content_element>
        <participant />
      </content_element>
    - <text_element faceIndex="5" alpha="156" r="99" g="169" b="204">
      - <text_chunk doUnderline="true" sameHref="true"
          commonHref="http://news.bbc.co.uk/2/hi/americas/3762591.stm">
          <text_token string="Aristide" />
          <text_token string="departs" />
          <text_token string="for" />
          <text_token string="South" />
          <text_token string="Africa" />
        </text_chunk>
      </text_element>
    </media_element>
  - <visual>
      <extent x="336" y="165" width="94" height="56" />
    </visual>
  </collage_element>
```

FIG. 9. XML Representation of an Information Sample.

Figure 9 shows the XML representation of an information sample on screen. The example is of a text information sample. `containerURL` represents the container from where this text is derived from. The sub-element `text_element` represents the actual text of the information sample. `faceIndex`, `alpha`, `r`, `g` and `b` attributes in the `text_element` sub-element represent the visual attributes of this information sample. The sub-element `visual` represents the size and position of the sample on screen. The sub-element `participant` stores the interest state of this information sample. In this example, there was no interest expressed on this sample, so the element is empty.

The option of saving the current information space state is provided in the "File" menu in combinFormation menu-bar. The "File" menu contains the options "Save" and "Save As", which work the same way as in normal applications. "Save" option saves the information space with the current file name and "Save As" provides an option to save

the current information space with another file name. As shown in the Figure 10, the Save dialog box also contains a checkbox with the label “Save History” on its side panel. If the user checks this checkbox, the complete history of the information space will be saved; otherwise just the current state is saved. We cover the details of history representations of information space in section 5.5.

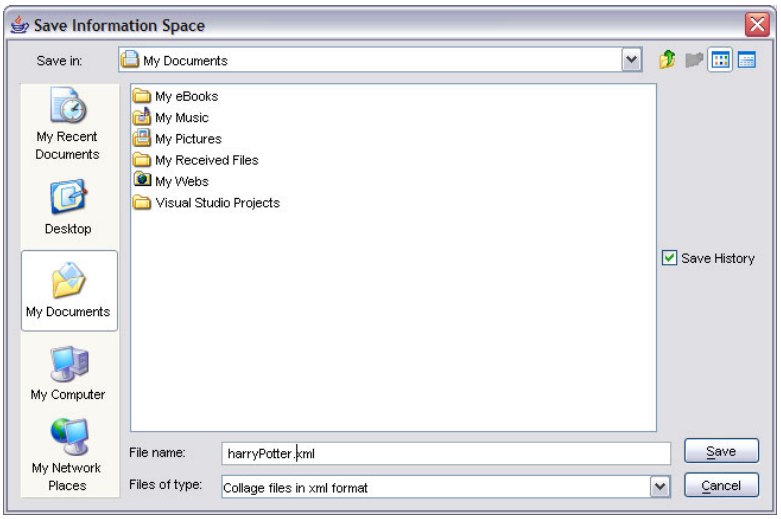

FIG. 10. Save Dialog Box in combinFormation. A checkbox is provided for an option to store information space with or without history. [Kerne and Interface Ecology Lab 2004]

5.4 Opening a Saved Information Space

The information spaces saved in the form of XML can be opened by the user to resume the information space session from the point where s/he left the previous time. The reverse translation mechanism for translating XML to Java, developed in the ecologylab.xml translation framework allows the program to convert the XML

representation in the saved information spaces into functional Java data-structures. The program builds the information space, which is visually the same as when the user saved last. The XML representation also stores the model of the user formed by the program in her/his last session, as a result of which the program is able to start from the same point in terms of the user model as well. In the next two sections we describe how an information space saved locally and an information space published on the web is opened using combinFormation.

5.4.1 Opening an Information Space Saved Locally

combinFormation can be started up with a blank information space. After this, just like in normal applications, the user can use the "File" menu and point to the saved file as shown in Figure 11. By looking at the header of the saved file, the program determines if the saved file contains the complete history of the session, or just a particular state of the information space. If the file contains the history, the complete session is restored; otherwise the saved state is restored. The JPEG file saved while saving the information space is used to display a preview of the information space in the side panel of the Open File dialog box. There is a check box with label "mix with existing"; if this is checked, the opened information space is merged with the current one; otherwise the current information space is closed and the contents of the space being opened are brought to the screen.

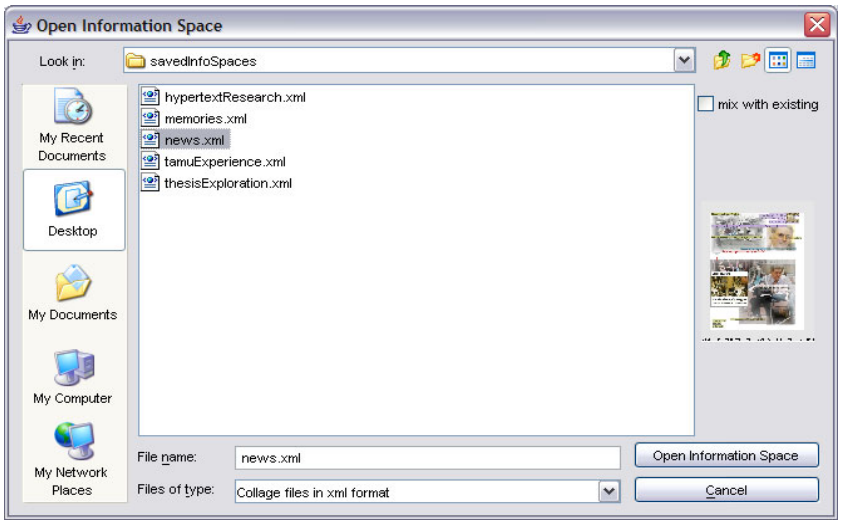

FIG. 11. Open Dialog Box in combinFormation. The side panel shows a preview of the selected file. [Kerne and Interface Ecology Lab 2004]

5.4.2 Opening an Information Space Published on a Web Server

We have added the capability to open information space from the web via HTTP, in addition to opening from the local file system. This is done by adding a publisher's mechanism that enables integration of these information spaces into regular web sites via the standard `href` tag. The mechanism consists of a new, fixed information space seeding page [Kerne et al. 2003]. References to it include the URL of published information space, and other optional metadata, as arguments.

For example an information space is saved with the following URL, `http://csdl.tamu.edu/collagemachine/launch/regenerate.html?metadocument=http://mrl.nyu.edu/andruid/perseusIntlChildrens.xml&start=play`. On clicking the URL, the user is brought to a standard startup page, which offers options such as the size of the information space visualization

window. The URL arguments and the options, which are processed via client-side JavaScript, are among the runtime parameters passed to the combinFormation applet. An optional publisher's runtime argument `start` determines whether the published information space is started as dynamic and evolving (playing), or as initially static (paused), giving the user time to see the visualization exactly as designed by the previous information space author. In either case, the user can use the tape recorder metaphor controls to switch between active play and static paused after the information space is restored.

5.4.3 Problems in Opening a Saved Information Space

Some surprising complications arise in regenerating the visual composition from the saved information space. The fact that the web is unstable and that the containers and images may suffer 404 Not Found errors, is to be expected. What is more complex is that, since download operations need to proceed in parallel, samples may not arrive in their original visual stacking order. Currently, we just let attributes from the model sort the stacking order out dynamically, as it would in the course of the program's usual, one sample at a time, operation.

5.5 History Representations of the Information Space

The representation for storing the complete history of the information space over time is inspired from the MPEG format for storing movies [MPEG]. The history representation consists of a set of *keyframe*s and *operations*. Each keyframe represents a full snapshot of the information space at a particular point in time; it is identical to the saved representation of the state of an information space at one moment. Keyframes are

emitted periodically. Interspersed between the keyframes are operations, which are the "deltas" that have occurred since the last full snapshot. The operations include those performed by the user, and those performed by the program.

To restore the state of the information space from time *t*, the program restores the last keyframe before *t* and then restores the operations until *t*. We note that, any operation cannot be restored until the last keyframe before that operation is restored. This is because the operations are performed on the existing information samples on screen and to restore those samples, the last keyframe must be restored. Once the last keyframe is restored, the samples are present on screen and operations can be performed on them.

This representation gives the program the ability to restore the state of the information space from any point in its timeline, while balancing space efficiency and time efficiency. Space requirements are optimized by keeping delta operations interspersed between periodic keyframes instead of just keeping a series of keyframes, which are have much more space requirements compared to delta operations. Restoring the last keyframe and then restoring the delta operations is much faster than restoring all the operations from start to the current time. This gives reasonable time efficiency.

In order to achieve the above representation, the program must emit keyframes at periodic intervals, and in-between the keyframes, it must emit the operations performed by the user and the program itself. The interval at which the keyframes are emitted can be in terms of time or in terms of number of operations. In other words, a keyframe can be emitted after every fixed amount of time or after every fixed number of operations.

We have chosen to emit keyframes after every fixed number of operations because of the non-linearity of combinFormation timeline and the fact that jog-shuttle slider represents "mixed-initiative operations" rather than time. The details of the semantics of the jog-shuttle slider and the timeline representation are discussed in section 4.1.

Adaptive algorithms have been applied in the past work to determine which snapshots should be stored [Kurlander and Feiner 1990] so that users can easily notice the transition and the change between two consecutive snapshots. We have taken a simpler approach. In the current implementation, a keyframe is emitted after every 25 operations. We arrived at the number 25 after doing trials observing the difference between two consecutive keyframes. The observation done during the trials was that the difference in terms of visual appearance of two consecutive keyframes should be such that users can see the transition from one keyframe to the next and still notice the changes that have occurred between the two keyframes.

The ecologylab.xml translation framework is used for emitting the keyframes and the operations in the form of XML from the corresponding Java structures. In addition to the XML form of the keyframe, the program also emits a small preview image file (245*325 pixels) for each keyframe, though not a full-sized JPEG, as in a full save. These are used to provide the periodic visual previews of the state of the information space to the user while s/he is traversing the history using the jog-shuttle history slider (See Section 4.1.2). The facilities developed in the ecologylab.xml translation framework also makes it easy and convenient to restore the state of the information space at any time from the XML representations of the keyframe and the operations. The

implementation details of how this representation is saved and used to restore states are described in the next section.

5.5.1 Memory Mapped History Represenation

Storing the complete set of keyframes and operations consumes a large amount of memory. The amount of memory needed depends on the number of operations that the user performs on the information space. The more the number of operations, the more is the required memory to save them. For a typical four-hour combinFormation session, the size of the XML data is approximately 150 MB, and the JPEG previews are approximately 20 MB.

If the program keeps all this history data in main memory, it will quickly exhaust reasonable resources. If the program stored this data on the disk using normal file-system mechanisms, the program would need to break the history down into chunks, which would be expensive to read and write, because the program will need to seek back and forth across the disk to restore the keyframes and operations every time the user wants to move to a chronologically distant state of the information space.

To overcome this problem, we use the technique of memory-mapped of files [Stevens 1998]. This facility, which has been part of Berkeley Unix for decades, is provided by Java 1.4 in the package `java.nio` by a class called `MappedByteBuffer`. The complete description of how this technique is employed for storing the history of information space is descried below:

The program first opens an empty `RandomAccessFile` on the local disk in read-write mode and gets a `Channel` to that file. It then creates a

`MappedByteBuffer` using the `Channel` obtained. This buffer is a memory map of the complete file. A very huge file size is chosen because we cannot anticipate the size of the history file at the beginning of the combinFormation session. The size of the history file is directly proportional to the session time. Just to make sure that there is enough capacity in the buffer, the initial size given is of the order of several hundreds of megabytes. Thus, the history mechanism requires this much free disk space on the user's computer.

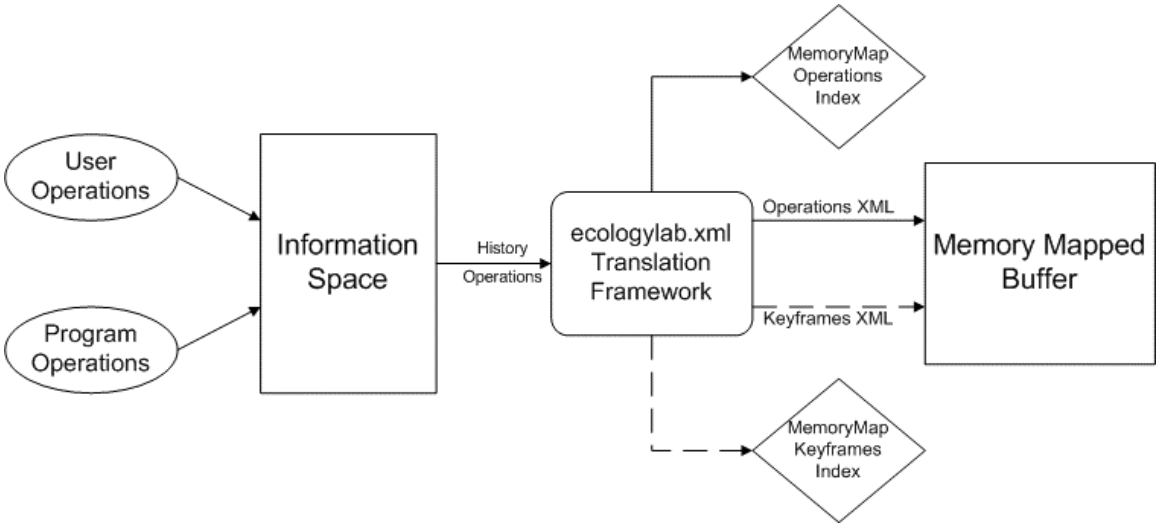

FIG. 12. Logging History Operations and Keyframes to Memory Mapped Buffer.

As mentioned before, the history of the combinFormation information space is stored similar to MPEG format. It is a sequence of keyframes and operations with 25 operations between every two successive keyframes. After every keyframe or operation, the XML corresponding to it is generated with the help of the ecologylab.xml translation

framework and is stored in the memory map buffer. Two separate indices are maintained in the memory indicating the position of the keyframes and the operations in the memory map buffer respectively as shown in Figure 12. Each of these indices is a hash map, which contains the timestamp of the keyframe/operation as the key and the information about the keyframe/operation as the value. The information maintained in the value consists of the starting byte offset value of the keyframe/operation in the memory map buffer, and the number of bytes used to represent that. In case of keyframes, the information also includes the file path of the corresponding JPEG file of the keyframe. It is necessary to store the path of the JPEG representing the keyframe, because the program needs to display this picture in the preview window, when the user moves to the location of this keyframe using the jog-shuttle slider.

The memory map index is used for restoring the keyframe or the operation when the user is moving across the timeline of the information space. As the user goes back and forth in the timeline, the keyframe/operation that needs to be restored is determined based on the position of the knob on the slider. The program then retrieves the keyframe/operation from that point in time using the memory map index. Now from the bytes retrieved from the memory map, it constructs an XML string and creates a DOM object from the XML. This DOM object representing the keyframe or operation is then translated by using our XML translation framework to the functional keyframe or the operation of the information space. The complete process of retrieving a keyframe from memory and displaying it is depicted Figure 13.

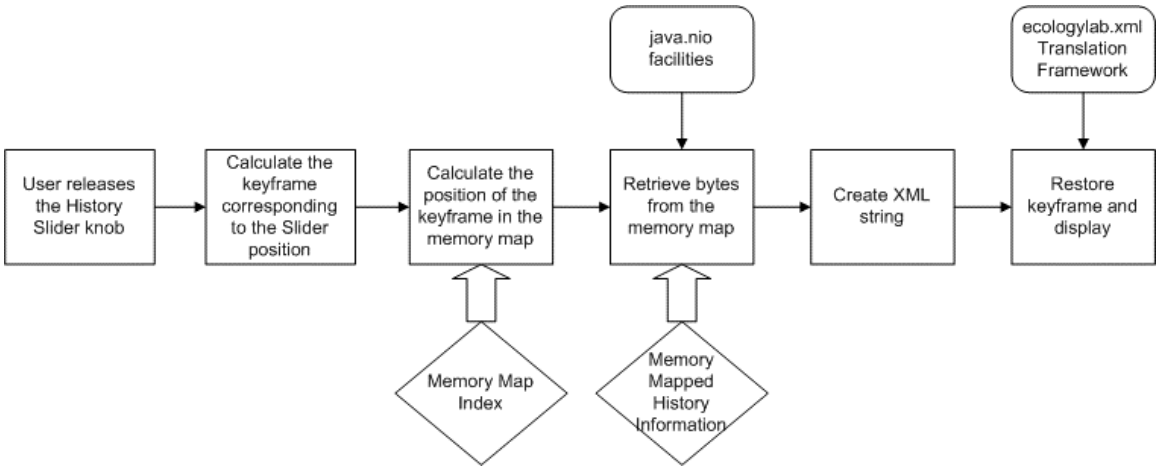

FIG. 13. Retrieving a Keyframe from Memory Map and Displaying It.

Overall, the memory mapping mechanism makes the history mechanism scalable to very large history sessions. It eliminates the main-memory space constraint for storing all history data, and provides quite acceptable performance for large histories, as we noticed during our user studies.

5.5.2 Caching Keyframe Image Files to Enable Fast Display during Timeline Traversal

As mentioned at the beginning of the section 5.5, the program saves JPEG image files of the keyframes, representing the snapshot of the information space, at regular intervals. These image files are used to display the keyframe previews when the slider is moved on the timeline. The real-time performance of these retrieval operations is critical, because otherwise, the previews that the user sees while dragging the jog-shuttle slider will not arrive quickly enough to actually assist the user in understanding the timeline.

The performance of simply retrieving each JPEG keyframe file from the disk and displaying it when the user moves the jog-shuttle is not sufficiently fast for real-time interaction. We want the keyframes to display instantly, as the jog-shuttle is dragged across the timeline, to enable the user to rapidly observe and compare different states, decide what is important, and which state to go to. In order to make sure that the preview appears rapidly, as the slider is moved, we have implemented an algorithm that caches keyframes in main memory. We allocate a fixed number of "bins" in the main memory, which store the cached keyframes. At any point in time, all the bins together have a set of keyframes representing the complete session. For example, if the total number of keyframes currently in the session in n, and we have k bins in the cache, then the keyframes are spaced at n/k distance. An illustration can be seen in the following discussion. Let us assume that the total number of bins in the cache is 8, i.e. $k = 8$.

When total number of keyframes in the session is $n = 8$, the bins contain:

0	1	2	3	4	5	6	7

When total number of keyframes in the session is $n = 16$, the bins contain:

0	2	4	6	8	10	12	14

When total number of keyframes in the session is $n = 24$, the bins contain:

0	3	6	9	12	15	18	21

When the user drags the slider, the program calculates the keyframe corresponding to the location of the slider, and retrieves the closest keyframe that exists in the cache. If a close match is not present in the cache, the program retrieves that keyframe from the disk, displays it and then caches it for future. This caching mechanism ensures that displaying of keyframes is almost instant, when user drags the slider and thus making it a pleasant experience for the user to look through different keyframes quickly and arrive at the desired location in the timeline.

5.5.3 Saving the History File

Similar to saving the state of information space at one moment, the user may want to save the information space with the complete history. This can be done by checking the "Save History" option presented in the Save Dialog Box, which was illustrated in Figure 10. To save the information space with history, the program needs to save all the keyframes and the operations to the required history file in the disk. To achieve this, the program saves the complete memory map buffer to the history file on the disk as requested by the user. Equally important is saving the memory map index. This is required because when the history file is opened later, the program still needs to be able to retrieve any particular keyframe and/or operation from the complete history information, which would not be possible without the memory map index. Thus, the program appends the memory map index to the end of the history file. The location of the memory map index in the history file also needs to be stored so that the memory map index can be retrieved from the file. So the location of the index is stored at the very end; after the memory map itself. In addition to this, the program also stores all the image

files emitted for each keyframe to the same folder as the XML file so that they can be used for providing previews when this saved file is re-opened later.

The overall history file looks as shown in the following figure:

```
<?xml version="1.0" encoding="US-ASCII" ?>
- <history>
  + <keyframe timeStamp="3124">
  + <ops>
  + <keyframe timeStamp="21991">
  + <ops>
  + <keyframe timeStamp="31475">
  + <ops>
  + <keyframe timeStamp="44223">
  + <ops>
  + <keyframe timeStamp="48139">
  - <memory_map_info>
    + <keyframes_index_set>
    + <operations_index_set>
    </memory_map_info>
    <mapPointer>00000000000000146723000000000000000011915</mapPointer>
  </history>
```

FIG. 14. XML Representation of an Information Space with History.

We can see from Figure 14 that the XML file containing the history of the information space consists of keyframes, operations and the memory map information. `keyframe` is exactly similar in structure to the XML file representing an information space without history, which was discussed in section 5.3.1. `ops` is a set of operations performed on the information space between two keyframes. The `memory_map_info` contains the index of the keyframes and the index of the operations. The `mapPointer`

contains the starting location of the keyframes index and the operations index respectively in the memory mapped buffer.

An example of the XML structure for each operation is depicted in Figure 15.

```
- <collage_op action="remove" time="39447">
  - <collage_element id="http://graphics7.nytimes.com/images/2004/05/30/sports/31indi3.jpg"
      containerURL="http://www.nytimes.com" historyNum="1">
    - <media_element>
      - <content_element>
          <participant />
        </content_element>
        <image_element hasTransparency="true"
          url="http://graphics7.nytimes.com/images/2004/05/30/sports/31indi3.jpg"
          href="http://www.nytimes.com/2004/05/30/sports/othersports/31INDY-
          CND.html" />
      </media_element>
    - <visual>
        <extent x="42" y="55" width="184" height="246" />
      </visual>
    </collage_element>
  </collage_op>
```

FIG. 15. XML Representation of an Operation on Information Space.

We note from Figure 15, that the XML representation for an operation consists of an attribute “action”, which indicates the type of the operation (add, remove, cut, grab, etc.) and a time stamp which indicates the time at which the operation was performed. The rest of the data is about the information sample on which the operation was performed.

5.5.4 Opening a Saved History File

The user can open a saved information space with history using the Open Dialog Box described in section 5.4.1. When the user supplies the history file to the Dialog Box, the program checks the header of the file to determine whether it contains the history

information or not. If it does not contain the history information, the file is opened as described in section 5.4. When a file containing the complete information space history is being re-opened in a new session, the program first needs to build the memory map index. Since the location of the memory map index in the history file is stored at the very end, the program first retrieves this information about location, goes to that location and fetches the bytes representing the memory map index. From this index, the XML form of last keyframe is retrieved and is restored to the corresponding Java structures using the reverse translation provided by ecologylab.xml translation framework. This is the state of the information space while saving because, just before saving the space, a keyframe is emitted. Thus we see that as a result of the memory map index, ecologylab.xml translation framework does not need to load the entire saved XML into a DOM for building Java structures, which will require prohibitive amount of memory for large histories. Instead, the keyframes and operations are restored from the memory map in an on-demand basis by the ecologylab.xml translation framework. In other words, the DOM is built only for individual keyframes and operations, as and when they are required to be restored.

6 EVALUATION

A usability study was conducted to evaluate the time travel features in combinFormation implemented during this work. Standard interaction design evaluation techniques as described by Preece, Rogers, and Sharp [2002] were used. We were interested in evaluating the aspects specific to the history mechanism as well as the overall changes in the experience of the user using combinFormation as a result of the history mechanism. The following points describe the information that we wanted to collect from the evaluation:

- Do users use the history of the information space?
- Can the users understand how to effectively use the visualization and interaction mechanisms offered by the software for history traversal and manipulation? In other words, is the user-interface clear and intuitive to use?
- What is the overall difference in the experience of the users using combinFormation with and without history interfaces?
- Do users use the non-linear history editing features offered by the latch tool provided in the software?

The study questionnaires were designed to get answers to the questions described above. The answers obtained from the participants of the study will help us understand the usefulness of the history interface. It will also give us feedback on ways to improve the interaction design and efficiency of the different algorithms used in the software for implementing the history mechanism. The details and the results of the study are described in the following sections.

6.1 Study Protocol

The study was conducted in the form of an information space (interactive poster) design competition. Students from Visualization Lab in the TAMU Architecture Department were invited to participate in this competition. Call for participation was circulated in the lab through email to the Lab mailing lists and fliers posted in the Lab. The first prize to be rewarded to the winner of the design contest was an amount of $100. The reasons for conducting this user study in the form a design competition are explained below:

- Validation of results

  As the users engage in finding information relevant to their research, this will provide a real-world usage scenario for the software. The results obtained will be more authentic compared to the mock usage scenario created in more typical experimental methodologies.

- Motivation for the participants

  We believe that the participants will have extra motivation due to the competitive environment created by this form of study and the incentive provided to the winner(s) in terms of monetary value and public recognition. The form of this competition is similar to that of others that they may engage in, such as designing a poster for taking it to SIGGRAPH or other design venues.

- Educational benefits

  The software provides the participants the opportunity to explore their topics of interest, with a focus on the thesis. This will help them in finding information

and discover ideas. For the participants who already have a thesis topic, this will facilitate the development of ideas, and provide new forums for obtaining feedback from the community and presenting ideas to the public. Additionally, some of the participants may choose to enter this interactive poster to national and international design exhibitions, such as SIGGRAPH.

6.2 The Task

Subjects were asked to design a poster clearly representing either their thesis topic, or another academic project of interest to them. The poster must be in the form of a combinFormation information space. The subjects were instructed that the posters should consist of a composition of text and image samples from web sites. They were also told that these web sites should include source materials that they are referencing in defining their thesis (or other topic). They could, if they wished, also include materials from their own web site.

The task was split in two stages

- In stage 1, participants worked on the version of the software without the history interface.
- In stage 2, participants worked on the version of the software with the history interface.

About half the participants did the experiment in the above order, while the other half of the participants did it in the reverse order. That is, they did stage 2 first and then did the stage 1. Participants were required to fill out one pre-questionnaire and two post-questionnaires. One post-questionnaire had questions corresponding to the interface

without history, while the other one had questions corresponding to the interface with history.

Each participant was involved in the following set of activities:

- Introduction to the project and motivation for this study, presented by the investigator and signing of release form (5 minutes)
- Participant's filling out of the pre-experiment questionnaire (max 5 minutes)
- Training on how to use the software, presented by the investigator (15 minutes)
- Participants using the software to get a feel of how it works (max 15 minutes)
- Participants evaluating the software in stage 1 (max 30 minutes)
- Participants filling out the post-experiment questionnaire A(max 10 minutes)
- Participants evaluating the software in stage 2 (max 30 minutes)
- Participants filling out the post-experiment questionnaire B (max 10 minutes)

6.3 Data Collection

Data was collected through questionnaires and invisible automated logging of software usage events. Participants filled out a questionnaire at the end of each stage of the task. They filled out a set of questions giving feedback about their experience after finishing the task using the software in stage one. They also filled out another questionnaire after doing the activity in the second stage of the task. Some of the questions were common in both the questionnaires to obtain direct comparison of the two interfaces. Other than the common questions, there were some particular questions about the history interface in the stage in which user was using the software with the history features. While the users were engaged in the task, the software also logged the

usage activity of the user, which included the different operations that were performed by the program as well as the user, the time for which they were performed and the different tools that were used by the user. This logging is achieved using the ecologylab.xml translation framework. Each operation has a corresponding Java representation when the program is running; this representation is emitted in the XML form by the translator after every operation performed in the software either by the program itself or by the user. This XML data has all the information related to each operation such as time when it was performed, the operation type, the tool that was used to perform the operation and the information sample, if any involved in the operation. The logged data in the form of XML is easy to mine using any XML parser and gives us all possible information that can be analyzed to extract meaningful conclusions. Some of the information that we can get using this log that is not possible to obtain using usual questionnaires is the frequency of usage of any particular tool in the software and the usage pattern of different tools in the software.

6.4 Results and Discussion

The results obtained from 6 subjects indicated that the users do find the time travel features including the jog-shuttle history slider and the latch-tool useful while making compositions using combinFormation.

6.4.1 Quantitative Results

According to the data obtained from the usage log files, we have calculated that on an average session of 26 minutes using combinFormation to make compositions, the

users make use of the jog-shuttle history slider about 2.5 times. This translates to about once in ten minutes. The reasons for using the jog-shuttle features to go back in the timeline are listed below:

- Go back and start over from a previous point in time.
- Go back, latch a few samples and jog-back to the current time.
- Go back, latch a few samples and start over from a previous point in time.
- Go back and forth in the timeline and return back to the current time.

We make a few observations based on the above results. At the normal speed of information space recording, 10 minutes is a long enough time for the information samples to start disappearing from the space if the user has not expressed interest on it. In this case the users went back in time and retrieved those disappeared samples from a previous point in time using the latch tool. 10 minutes might also be a long enough time that the program might either start wandering from its original search focus or start getting irrelevant restuls, if the user has not expressed her/his interests properly or if the program is running out of contents for the initial search topics. In this case, the users used the jog-shuttle to go back to a previous point in time when the information space was at a state which the user liked. The users then expressed positive and negative interest on the appropriate information samples and started recording again from that point. This enabled them in changing the direction of the web-crawler and thus resulting in information space bringing up more relevant samples. Some users just jogged back and forth comparing different states using the preview and ended up coming back to the current time.

We describe a few other comparative results obtained from the questionnaires, which compared the experience of the user doing the activity with the software with and without time travel features. The users were asked to rate their answers to questions on a 5-point scale of 0 through 4, with 0 representing the minimum and 4 being the maximum.

- **In-Control of the information space**: Users were asked how much in control of the information space they felt. The average for users using the software without time travel features was 1.4, while that with the time travel features was 2.0. This indicates that users have more control to the space when they have access to the time travel features of the tape-recorder controls.
- **Success at doing the activity**: The users were asked the level of success of their task. The average for users using the software without time travel features was 2.4, while that with time travel features was 2.8. This indicates that the users felt slightly more successful doing their activity with the interface providing the time travel features.

6.4.2 Qualitative Results

There were a few questions in the questionnaire which asked the users to compare their overall experience in using the software with and without time travel features. The main results obtained from these questions are:

- When the users had access to the time travel features, they were not afraid of losing any information sample and could concentrate more on making the composition.
- Some users responded that using a combination of the timeline and the latch tool, they could get back information samples that were deleted by accident. We note that this was also possible using the undo command but users tend to not use this command as it involves repeated hitting of keys or repeated clicks on the menu.
- The users also said that using the time travel features, they could prevent the program from going off-topic. This is consistent with the quantitative result explained in the above section that they felt more in-control of the space.
- Some users found the use of play forward and record buttons confusing. This could be attributed to the fact that in the version of the software without time travel features, play is used to generate new samples. In the newer version with time travel features, play is used to play forward the already occurred sequence of operations of the information space. To generate new information samples, record button must be used in the newer version. We expect that if the user is using the newer version without ever looking at the old interface, this confusion may be avoided.

7 CONCLUSIONS AND FUTURE WORK

The ability to go back and forth in the information space timeline enable the users to explore the information space through its series different states over time. It helps the authors and the readers of the information space to experience the complete process of information collecting session, rather than just the final state. It heps the users to compare different states of the information space and see the emerging information resulting from the process of information collecting.

We have developed interactive mechanisms to enable the users to access the information space timeline. We have extended the tape-recorder metaphor controls of the combinFormation information space in order to enable the users to operate the timeline. The most important new control is the jog-shuttle history slider, which can be used to rapidly seek through the timeline and browse different states of the information space. The slider also gives visual previews to show the different states, while it is being dragged. The new controls also include the play back and forward buttons to repeat the already occurred series of operations on the information space in a forward or backwards fashion.

We have also explored a few forms of non-linear history manipulation, which could include maintaining multiple branches, editing past history actions and moving information samples across time. We provide the implementation of only the third case, which is moving the information samples across time by fixing its position. We have introduced the latch tool based on door-latch metaphor, which enables users to latch

information samples and fix their position across time. An information sample when latched overrides all other history operations that could have changed its position. Using this feature, users can move the information samples that existed in the past to the current time.

There are a number of directions where this work could go from the current state in order to enhance the time travel features. We list a few of them below:

- More sophisticated algorithms can be implemented for determining the keyframe intervals so that keyframes with insignificant changes are not emitted. This will enable the users to notice the changes across states quickly and facilitate better comparison. A simpler thing that can be implemented is to provide a preference using which the users can set their own interval for emitting the keyframe.
- In the current scope of work we have provided a solution to only one form of non-linear history manipulation. More forms of non-linear history can be provided, which include but are not limited to maintaining multiple branches and editing past history actions.
- Facilities can be provided to search the timeline when the user is looking for information samples not present in the current state, but that might have existed at a previous point in time.
- The jog-shuttle history slider can be made to zoom in and zoom out so that it represents only a particular region of the timeline. This can help the users to focus and explore particularly interesting regions of the information space timeline.

- Since the program emits a JPEG image file for each keyframe, a panel can be provided to the user in which the users can see all the keyframes at once. This will facilitate side-by-side comparison across states since the user can see all the intermediate states at once without scrubbing the jog-shuttle history slider. The problem in this is that when the number of keyframes becomes large, presenting all of them without requiring the users to scroll through pages becomes a problem because of the limited screen real estate. Advanced user interface techniques such as Zoomable User Interfaces [Bederson et al. 2000] can be employed to lessen this problem.
- The information space can be used as a new kind of visual medium. More forms of non-linear editing capabilities to the timeline can be added to add more ways of manipulating the medium. This form of temporal medium can be further developed as a temporal presentation medium with non-linear editing capabilities.

REFERENCES

AYERS, E., AND STASKO, J. 1996. Using graphic history in browsing the World Wide Web. In *Proceedings of the 4th International World Wide Web Conference*. O'Reilly Associates, Cambridge, MA, (See http://www.w3j.com/1/ayers.270/paper/270.html/.)

BEANML See http://www.alphaworks.ibm.com/formula/bml/. Accessed Date: June 2004.

BEDERSON, B.B., MEYER, J., AND GOOD, L. 2000. Jazz: An extensible zoomable user interface graphics toolkit in Java. In *Proceedings of UIST 2000, ACM Symposium on User Interface Software and Technology*. ACM, New York, 171-180.

BURKE, M. 1999. *Organization of Multimedia Resources*. Gower, Hampshire, UK.

BUSH, V. 1945. As we may think, In *Atlantic Monthly*. The Atlantic Monthly Group, Boston, MA, 101-108, (See http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm/.)

CATLEDGE, L., AND PITKOW, J. 1995. Characterizing browsing strategies in the World Wide Web. In *Proceedings of 3rd International World Wide Web Conference*. Elsevier North-Holland, New York, 1065-1073.

COCKBURN, A., GREENBERG, S., MCKENZIE, B., SMITH, M. AND KAASTEN, S. 1999. WebView: A graphical aid for revisiting Web pages. In *Proceedings of OZCHI '99, Australian Conference on Human Computer Interaction*. Wagga Wagga, Australia, See (http://www.cpsc.ucalgary.ca/Research/grouplab/papers/1999/99-WebView.Ozchi/Html/webview.html)

COCKBURN, A., MCKENZIE, B., AND SMITH, M. J. 2002. Pushing back: Evaluating a new behavior for the back and forward buttons in Web browsers. In *International Journal of Human Computer Studies*. Academic Press, Orlando, FL, 397-414.

DOM See http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/. Accessed Date: June 2004.

FURUTA, R. SHIPMAN, F., MARSHALL, C. BRENNER, D., AND HSIEH, H. 1997. Hypertext paths and the World-Wide Web: Experiences with Walden's Paths. In *Proceedings of 8th ACM Conference on Hypertext*. ACM, New York, 167-176.

GOSLING, J., JOY B., AND STEELE, G.L. 1996. *The Java Language Specification*. Addison-Wesley, Reading, MA.

GREENBERG, S., AND WITTEN, I. 1988. How users repeat their actions on computers: Principles for design of history mechanisms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 171 – 178.

HIGHTOWER, R., RING, L., HELFMAN, J., BEDERSON, B., AND HOLLAN, J. 1998. Pad prints: Graphical multiscale Web histories, In *Proceedings of UIST 1998, ACM Symposium on User Interface Software and Technology*. ACM, New York, 58 – 65.

Horvitz, E. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 159 – 166.

JAXB See http://java.sun.com/developer/technicalArticles/WebServices/jaxb/. Accessed Date: June 2004.

JOX See http://www.wutka.com/jox.html/. Accessed Date: June 2004.

KBML http://koala.ilog.fr/kbml/. Accessed Date: June 2004.

KERNE, A. 2001. CollageMachine: A model of interface ecology, *PhD. Dissertation*, New York University, NY.

KERNE, A., AND INTERFACE ECOLOGY LAB, 2004. See http://ecologylab.cs.tamu.edu/combinformation.

KERNE, A., AND SMITH, S.M. 2004. The information discovery framework. *To appear in the Proceedings of Designing Interactive Systems.* ACM, New York.

KERNE, A., AND SUNDARAM, V. 2003. A recombinant information space. In *Proceedings of Computational Semiotics in Games and New Media (CoSIGN).* Middlesbrough, England, 48-57.

KERNE, A., KHANDELWAL, M., AND SUNDARAM, V. 2003. Publishing evolving metadocuments on the Web. In *Proceedings of 14th ACM Conference on Hypertext.* ACM, New York, 104-105.

KLEMMER, S., THOMSEN, M., PHELPS-GOODMAN, E., LEE, R. AND LANDAY, J. 2002. Where do Web sites come from? Capturing and interacting with design history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 1-8.

KURLANDER, D., AND FEINER, S. 1990. A visual language for browsing, undoing, and redoing graphical interface commands. In *Visual Languages and Visual Programming,* Plenum Press, New York, 257-275.

LEE, A. 1992. Investigations into history tools for user support. *PhD Dissertation*, University of Toronto, Toronto, Canada.

LEVINE, J. R., MASON, T., AND BROWN, D. 1992. *Lex and Yacc (A Nutshell Handbook)*, O'Reilly Associates, Cambridge, MA.

LUESEBRINK, M. 1998. The moment in hypertext: A brief lexicon of time. In *Proceedings of 9th ACM Conference on Hypertext*. ACM, New York, 106-112.

MPEG ISO/IEC 14496-1, 12, 14:2003, Information technology - Coding of audio-visual objects -Part 1, Part 12, Part 14. See http://www.iso.org/.

NIELSEN, J. 1990. The art of navigating through hypertext, In *Communications of the ACM*. ACM, New York, Volume 33, Issue 3, 296-310.

PREECE J., ROGERS I., AND SHARP, H. 2002. *Interaction Design*, John Wiley & Sons, Indianapolis, IN.

REEVES, B. 1993. Supporting collaborative design by embedded communication and history in design artifacts. *PhD. Dissertation*, University of Colorado, Boulder, CO.

REKIMOTO, J. 1999. Time-Machine computing: A time-centric approach for the information environment. In *Proceedings of UIST 2000, ACM Symposium on User Interface Software and Technology.* ACM, New York, 45-54.

SCHRAEFEL, M.C., ZHU, Y, MODJESKA, D., AND WIGDOR, D. 2002. Hunter Gatherer: Interaction support for creation and management of within-Web-Page collections. In *Proceedings of 11th International Conference on World Wide Web.* ACM, New York, 172-181.

SHIPMAN, F., HSIEH, H., AIRHART, R., MALOOR, P., AND MOORE, J.M. 2001. The visual knowledge builder: A second generation spatial hypertext. In *Proceedings of 11th ACM Conference on Hypertext*. ACM, New York, 113-122.

SHIPMAN, F.M., HSEIH, H. 2000. Navigable history: A reader's view of writer's time, In *The New Review of Hypermedia and Multimedia*. Taylor & Francis Group, London, UK, Volume 6, 147-167.

STEVENS, R. 1998. *Unix Network Programming Vol. 2: Interprocess Communication*. Prentice Hall, New York.

TAUSCHER, L., AND GREENBERG, S. 1997. How people revisit Web pages: Empirical findings and implications for the design of history systems, In *International Journal of Human Computer Studies.* Academic Press, Orlando, FL, 97–138.

Weynand, D. 2004. *Final Cut Pro 4 – Editing Professional Video*, Peachpit Press, Berkeley, CA.

WOODRUFF, A., FAULRING, A., ROSENHOLTZ, R., MORRSION, J., AND PIROLLI, P. 2001. Using thumbnails to search the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 198 – 205.

XML See http://www.w3.org/TR/REC-xml/. Accessed Date: June 2004.

Supplemental Sources

KERNE, A. 1997. CollageMachine: Temporality and indeterminacy in media browsing via interface ecology, In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 297-298.

KERNE, A. 2000. CollageMachine: An interactive agent of Web recombination. *Leonardo.* MIT Press, Cambridge, Vol. 33, No. 5, 347-350.

KERNE, A. 2002. Concept-context-design: A creative model for the development of interactivity, In *Proceedings of the Fourth Conference on Creativity & Cognition, an ACM SIGCHI International Conference*. ACM, New York, 192-199.

KERNE, A. 2002. Interface ecosystem, the fundamental unit of information age ecology, *In Proceedings of SIGGRAPH02*: *Art and Animation*. ACM, New York, 142-145.

MARSHALL C.C, SHIPMAN, F.M., AND COOMBS, J. 1994. VIKI: Spatial hypertext supporting emergent structure. In *Proceedings of the ACM European Conference on Hypermedia Technology*. ACM, New York, 13-23.

PIROLLI, P., PITKOW, J., AND RAO, R. 1996. Silk from a sow's ear: Extracting usable structures from the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 118-125.

ROSENBERG, J. 1996. The structure of hypertext activity. In *Proceedings of 7th ACM Conference on Hypertext*. ACM, New York, 22-30.

SHIPMAN, F.M., AND MARSHALL C.C. 1999. Formality considered harmful: Experiences, emerging themes, and directions on the use of formal representations in interactive systems. In *Proceedings of Computer Supported Cooperative Work*, ACM, New York, 333-352.

SHIPMAN, F.M., MARSHALL, C.C. AND MORAN, T.P. 1995. Finding and using implicit structure in human-organized spatial layouts of information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, ACM, New York, 346-353.

SUCHMAN, L. 1987. *Plans and Situated Actions, The Problem of Human–Machine Communication*. Cambridge University Press, Cambridge, U.K.

TUFTE, E. 1990. *Envisioning Information*. Graphics Press, Cheshire, CT.

VITA

Name	Madhur J. Khandelwal
Permanent Address	Khandelwal Rice Mill, Murri Road, Gondia – 441601 Maharashtra, India
Education	B.E. in Computer Science, 2002 Birla Institute of Technology and Science, Pilani, India MS in Computer Science, 2004 Texas A&M University College Station, TX - 77801
Professional Experience	Undergraduate Intern, Motorola India Electronics Limited, Bangalore, India, January 2002 – June 2002 Software Developer in Test Intern, Microsot Corporation, Redmond, WA, Summer 2003
Conference Papers	"Publishing Evolving Metadocuments on the Web", In Proceedings of ACM Hypertext, 2003, 104-105. "Manipulating History in Generative Hypermedia", To appear in ACM Hypertext, Santa Cruz, CA, 2004.