

Visualizing Threaded Discussions

by

May-Li Khoe

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 22, 2000

[June 2005]

© Copyright 2000 May-Li Khoe. All rights reserved.

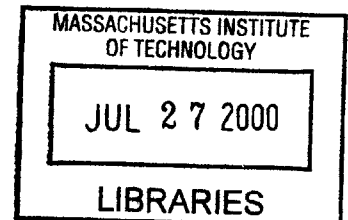
The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author _____
Department of Electrical Engineering and Computer Science
May 22, 2000

Certified by _____
Julie Dorsey
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

ENG



Visualizing Threaded Discussions

by
May-Li Khoe

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 2000

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The main goal of this project is to design a visualization and improved user interface for threaded discussion forums, a kind of software commonly interfaced through the web today. These forums contain discussion threads, also known as threaded discussions: groups of replies which, directly or indirectly, originate from one initial message. Current interfaces for threaded discussions provide only minimal functionality, and are often cumbersome to navigate. Despite the graphical nature of the World Wide Web, no visualizations or meaningful images are provided, and meta-data which could supply users with greater context is not displayed. Limited screen space is part of the problem - if too much information is presented in textual form, the screen becomes cluttered. Balancing this trade-off is an inherent problem with discussion forums, since it is a challenge to compress large amounts of textual data into a compact web page. A visualization of this discussion data with effective design and interactivity could be a step toward solving the problem. Our designs are aimed at helping users save time and energy while seeking information within discussion databases.

Thesis Supervisor: Julie Dorsey

Title: Associate Professor in the Departments of Electrical Engineering and Computer Science and Architecture

Table of Contents

ABSTRACT	2
TABLE OF CONTENTS	3
CHAPTER 1: INTRODUCTION	5
1.1 TERMINOLOGY	5
1.2 STATEMENT OF PROBLEM	6
CHAPTER 2: PROJECT OVERVIEW	8
2.1 GOALS OF PROJECT	8
2.2 OVERVIEW OF THE PROCESS	10
2.3 OBSERVATIONS	12
2.4 SUGGESTIONS	17
2.5 NARROWING DOWN THE SUGGESTIONS	21
CHAPTER 3: THE DESIGN OF THE VISUALIZATION	23
3.1 INTRODUCTION TO THE DESIGN	23
3.2 DEFINITIONS AND DESIGN PRINCIPLES	25
3.3 THE DESIGN PROCESS	29
3.4 BRIEF OVERVIEW OF LEVEL ONE	31
3.5 DESCRIPTION OF LEVEL TWO	35
3.6 BRIEF OVERVIEW OF LEVEL THREE	45
CHAPTER 4: EXPERIMENTING WITH IMPLEMENTATION	48
4.1 INTRODUCTION TO THE PROTOTYPE	48
4.2 DIRECTOR AS A FRONT END TO NOTES	49
4.3 NETWORK COMMUNICATION BETWEEN NOTES AND DIRECTOR	52
4.4 MANAGING THE DATA COMMUNICATED BETWEEN NOTES AND DIRECTOR	54
4.5 MANAGING THE DATA WITHIN DIRECTOR	59
4.6 CREATING SPRITES DYNAMICALLY	61
4.7 OVERVIEW OF THE IMPLEMENTATION	64
CHAPTER 5: DISCUSSION	70
5.1 RESULTS AND ISSUES	70

5.2 RELATED WORK	72
5.3 FUTURE DIRECTIONS	73
5.4 ACKNOWLEDGEMENTS	74
<u>REFERENCES</u>	76

Chapter 1: Introduction

1.1 Terminology

This thesis will focus on the graphical user interface for threaded discussion forums, a kind of software commonly interfaced through the web today. Threaded discussion forums are reminiscent of message boards, in which people can post new messages or reply to existing ones. Each new topic posted can be followed by a series of replies. A group of replies which, directly or indirectly, originate from one initial message, can be called a *threaded discussion*, or a *discussion thread*. Each message in a discussion thread is considered an *entry* in that thread. A discussion *forum* usually contains and provides the interface to these threads, usually grouping those with similar subject matter.

Throughout this paper, threaded discussion forums will be referred to interchangeably as discussion databases, or simply threaded discussions. Thread entries can also be called contributions, messages, or postings. The replies to a message can be referred to as its child entries, and the message from which a reply originates is called its parent entry. The descendants of an entry include its parent entries, and all of their parent entries, and so on, until there are no more entries with parents to trace back to.

This paper will also make many references to the term "meta-data". By general definition, meta-data refers to data about data. For example, meta-data can include data about data elements or attributes, (name, size, data type, etc), data about records or data structures (length, fields, columns, etc) and data about data (where it is located, how it is associated, ownership, etc.). Meta-data may include descriptive information about the

context, quality and condition, or characteristics of the data [1]. For our purposes, meta-data shall refer to all data related to threads and entries other than the main body of entry text.

1.2 Statement of Problem

Current interfaces for threaded discussions, both on the web and in Lotus Notes, are quite basic – they provide minimal functionality, and are often cumbersome to navigate. Users are generally presented with a purely textual interface, despite the graphical nature of the World Wide Web. There are usually no visualizations or meaningful images to help a user understand and navigate discussion forums or discussion threads. Thus most existing discussion databases do not possess the nearly same caliber of presentation as it has potential for.

At present, little context is provided in current forum interfaces, not in the form of either textual or visual information. Providing context helps users determine what they've read, what others have read, and what they might want to read next. Furthermore, it is common to navigate discussion databases using a text search, which often lands users at an entry in the middle of a discussion thread. Due to the lack of context normally accompanying the display of an entry, it is often cumbersome to find the rest of the thread, including the descendants and children of an entry. The inherent problem with browsing search results is that each document is no longer presented as part of a bigger picture. Providing easily accessible context for threads and thread entries could help alleviate this problem within discussion entries.

Another problem is that there is a limited amount of screen space for displaying the large amount of potentially useful information contained in threads and in forums. The result is a trade-off - it is difficult for a user to navigate a site that displays too little information at a time, but if too much information is presented in a purely textual form, the screen becomes too cluttered. Thus, most existing interfaces would not have space for much meta-data, even if it were collected and available.

As a result, much of the information that provides users with context, and a better idea of what they want to read, is not available at all. Providing discussion data in a meaningful and navigable manner could save people time and energy exploring threads that do not contain the information they are seeking. There is still much exploration still to be done about which information is more useful, and importantly, how it can be presented in a way that is helpful to users.

Chapter 2: Project Overview

2.1 Goals of Project

The visual interface we have designed aims to provide more than the current basic and textual interface, through both additional meta-data and visual interpretations of that data. In order for this to be effective, we searched for appropriate idioms of expression to represent the meta-data we chose for display. As stated earlier, this meant that we needed to design visual representations which people could identify with and understand quickly, once they learned how.

To do this, we needed to make use of the basic elements of design to ensure that attributes were emphasized when they were of potential interest to the user, and de-emphasized when they were not. It was also important to make use of the natural hierarchical structure of discussion databases, and consequently offer multiple macro and micro levels of data viewing. Since our environment was potentially interactive, we wanted to incorporate currently underutilized interaction strategies, allowing users to probe the visualization for more further data using their mouse input. These techniques will help, not only by providing the user with additional context, but by assisting users in learning the idioms of expression we propose.

Some amount of learning how to read our visualizations might be required on the user's part, although we were generally searching for representations which were intuitive and consistent. There is a trade off between ease of use and ease of learning: in this case we felt as if some amount of learning was acceptable, if afterward the user

found the interface easier to use. In other words, users might require explanations of our visual representations when they first view the interface, but after they have viewed it several times, it should be easy to understand what is going on at a quick glance.

The work in this project is targeted more at a business-oriented community, rather than considering the leisure use of discussion forums. Businesses strive to manage and utilize their knowledge assets, but are currently limited from using discussion databases as toward such a purpose, despite the amount of information stored in them. This is because discussion forums are limited by the inefficiency of their navigation. Goal-directed users need to be able to find and retrieve information from these databases without losing a lot of time to ineffective searching and reading. They need to be able to decide quickly which threads to examine, and which entries in a thread might be interesting. If these discussion forums can be reasonably navigable knowledge sources, rather than large stores of information which no one can efficiently wade through, then businesses will want to host more of them. Then these could be used by current and potential customers, business partners, and employees, and be kept afterwards as future references, becoming a valuable corporate resource.

In order to demonstrate our ideas to the business community, it was necessary to build a proof of concept prototype. The main purpose of a partial implementation was to illustrate that our design proposals actually had potential feasibility, and to test our visualization against some nominal amount of data. It also helped us to determine which development tools could be worthwhile in the realm of interactive data visualization, since narrowing large amounts of data into a compact, web-browsable multimedia presentation is no small matter. The creation of a prototype showed our theoretical ideas

in context, so that others in the development community could quickly see their value, more so than when they are only expressed in formal presentation. Thus a prototype was mainly to help demonstrate that our ideas were useful and feasible enough to continue researching and developing in the future.

Currently, there are limited idioms of expression for the attributes of discussion threads; those that do exist are mainly hierarchies, presented in an outline list. Thread and forum attributes are often abstract concepts, such as the relationship of replies, amount of activity in the thread, etc. Thus intuitive visual representation is a challenge – for example, there is no clear idea in most people’s minds of a picture that could effectively represent a reply.

Due to a lack of former work done in this field, there was a substantial amount of exploration required to develop idioms of expression for threads and their meta-data. While we experimented with various designs to see what worked and what didn’t, we were not aiming to discover a set of definitive symbols.

The scope of our subject matter is potentially overwhelming, given the length of time allotted. The amount of exploration needed in the design aspects alone could have been a project of immense size and duration. There are various issues around the implementation, security, privacy, and validity, described later in this chapter, which influenced our choices of what to design and implement. We concentrated on proposing an improved visualization and graphical interface to discussion forums on the web.

2.2 Overview of the Process

Designing a visualization for threaded discussion databases was a process which involved observation as well as design experimentation. For the first three months of the project, Hyun-Yeul Lee, a graduate student in Interaction Design at Carnegie-Mellon University, and myself, worked together with Paul Moody, a Design Researcher at Lotus Research. For the final three months of the project, there was a greater focus on exploring implementation, at which point only Paul Moody and myself were involved in the process.

To begin thinking about a visualization, we needed to have some basic knowledge about the kind of data we were trying to represent. We browsed many existing discussion database interfaces, and observing not only how they chose to present their data, but also what kind of data they were dealing with. It was important to see how their discussion databases were meant to be interacted with by users, to study the types of information being discussed, and to chart the resulting structure of the discussions. Doing this exercise helped us understand what sort of interface improvements could be made that would provide better navigational assistance to the average user. We also observed that a lot of information about thread activity was missing, information that could be extremely helpful to viewers who are trying to decide what to read next.

From our observations, we compiled a list of meta-data missing from current systems which we felt would be useful to provide. We also began to brainstorm about improvements possible to the existing interfaces, narrowing our focus to the visualization of the aspects we felt were within the scope of our project. It was also necessary to decide which meta-data would be immediately useful to explore. Since some of this information

had never been provided to users before, a certain amount of hypothesizing was necessary to this effect.

After choosing which meta-data to visualize, we began to design some visualizations. Our designs began to try and determine the right visual metaphors for the data in question - pictures or symbols which represented aspects of discussion entities in an easily comprehensible manner. The designs we came up with began with investigations of existing designs, from man and from nature, which served similar purposes, such as algorithms for organic growth, or existing methods for monitoring large amounts of data. Our ideas started out as hand-drawn sketches on paper or on whiteboards, and were further developed through iterations of redesign and critique. After deciding upon a particular visual representation of some data, it was also important to explore the types of user interaction possible with the new design. We began experimenting with how the design should react to mouse events and other user input, as well as dealing with issues such as extremes in quantity or space limitations. Each design for interactivity was also subjected to iterations of critique, redesign, and fine-tuning.

Eventually, the process of visual and interaction design needed to give way to some exploration in methods of implementation. So, we proceeded to investigate some of the tools which seemed potentially useful for an implementation. We then narrowed in on a part of the design and began implementing it.

2.3 Observations

In order to collect information about current discussion database interfaces and existing thread data, we went to sites such as delphi.com, deja.com (which provides a web interface to Usenet), ecircles.com, slashdot.org, and quickplace.com (a site offered by Lotus). All of these are examples of sites which host discussions. Another example of an interface to online discussions is that of Lotus Notes, which has its own discussion database design, which is browsable using the Notes client, or via the web.

Of the sites we examined, none of them offered any sort of visualization of thread structure. The ones that revealed structure did so via simple textual lists presented in a hierarchical structure, such as those shown in Figure 2.3.1

The screenshot shows a list of messages in a hierarchical format. The root message is 'Msg 1' by 'mitch_nete' dated 5/11/2000. It has eight replies: 'Msg 2' (Michael Valentiner=Branth, 5/12/2000), 'Msg 3' (Dave Roberts, 5/12/2000), 'Msg 4' (Frode Langset, 5/12/2000), 'Msg 5' (Christian, 5/13/2000), 'Msg 6' (robin, 5/14/2000), 'Msg 7' (Marko Teittinen, 5/15/2000), 'Msg 8' (Sarr J. Blumson, 5/15/2000), and 'Msg 9' (mitch_nete, 5/15/2000). The interface includes 'Messages 1-9 of 9 matches' and 'Page 1 of 1' at the top and bottom.

Message	Author	Date
» Msg 1 «	mitch_nete	5/11/2000
├─ Msg 2	Michael Valentiner=Branth	5/12/2000
├─ Msg 3	Dave Roberts	5/12/2000
├─ Msg 4	Frode Langset	5/12/2000
├─ Msg 5	Christian	5/13/2000
├─ Msg 6	robin	5/14/2000
├─ Msg 7	Marko Teittinen	5/15/2000
├─ Msg 8	Sarr J. Blumson	5/15/2000
├─ Msg 9	mitch_nete	5/15/2000

Figure 2.3.1. An example of a hierarchical list from deja.com

At the most macro level, the web forum interfaces generally offered a list of the major discussion areas available at their site. Occasionally, they also highlighted some of the most active discussion forums of the day, or chose particular topics which they thought were the “hot topics” of the day. An example taken from delphi.com is shown in Figure 2.3.2. The major discussion areas are listed at left, and some potentially interesting or active have been chosen and highlighted on the right.



Figure 2.3.2. The initial page displayed at delphi.com.

On many sites, once a particular subject area is selected, a page displaying another list of sub-areas of discussion, or a list of the threads in that subject area, is

shown. These threads are most often listed by the subject of their first or “root” entry, and are often accompanied by a number indicating their total number of posts. . Throughout the various interfaces we examined, other meta-data is occasionally displayed. For example, in some forums the author and/or date of the root entry is listed, while in others, the author and/or date of the latest entry are listed. On most sites, if the user browsing the site has never browsed it before, the threads will all be marked as “new”, showing that they had never been read by that user (see Figure 2.3.3). Once the user has read some of the messages in the thread, it is no longer marked new. In some interfaces, a number indicating the number of messages read and unread by that user is listed, although unread messages are often indicated as being “new”, even if they were posted before the user read other messages in the thread. This had the effect of misleading the user to believe that there are new posts to read, when in fact, there were only old posts that they chose to ignore earlier.

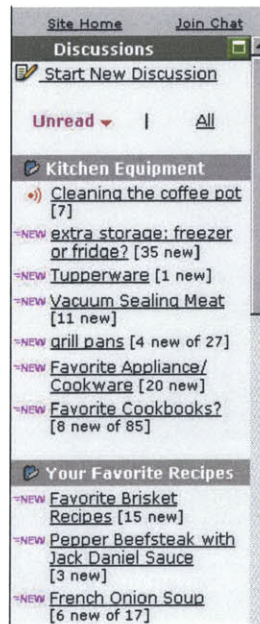


Figure 2.3.3. A list of unread threads marked as “new” at delphi.com.

When a thread's entry is selected for display, the actual author, date, time, and body of the entry is shown, as illustrated in Figure 2.3.4.

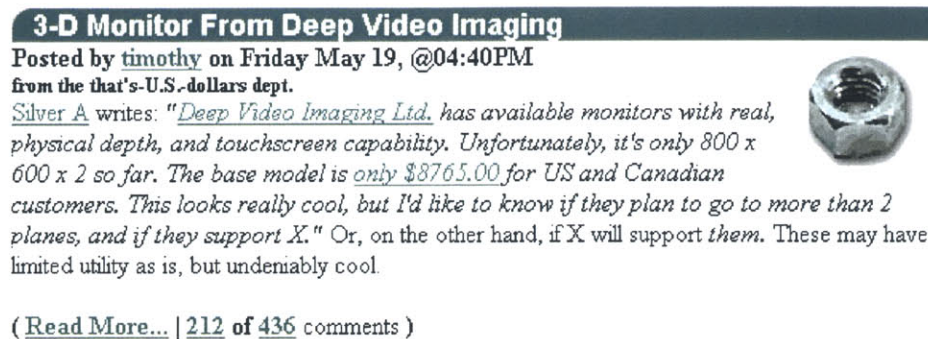


Figure 2.3.4. Example view of an entry from slashdot.org.

This is offered along with some very basic navigational tools: the user has to read one entry at a time, and click “next” or “previous” buttons on the page to continue reading consecutive entries in the thread. There might be an indication of what number an entry is in a thread, for example, number 33 out of 122. In *deja.com*'s interface to Usenet (the main body of discussion databases on the web), a basic list presentation of hierarchical thread structure is actually offered along with the display of the entry data, with the entry's position highlighted in the thread. However, when a viewer browses a particular entry which they have found using the search function, the hierarchical presentation is not provided. This is the one of the very few interfaces we found which provided thread context, let alone one which assisted navigation by displaying thread structure at the same time as entry data.

Even though we browsed hundreds of threads, we found that there were only two types of threads which occurred throughout the web forums we investigated. A

“problem” thread discusses problems, symptoms of problems, and solutions, and therefore is directed. An “opinion” thread does not contain entries which are directed at identifying and solving problems, but rather is open-ended, allowing users to voice their opinions about some issue.

We also observed that a large variability in thread size and structure existed throughout the forums we investigated. . There were some threads with only a few entries and a few users contributing to a coherent discussion, and there were other threads with hundreds of people participating and thousands of entries, with significant branching into tangents throughout. Some of this variability was due to the popularity of the hosting site or the thread topics. Of particular note, thread branching seemed to related to tools for entry authorship provided by the forum, which did not always clearly indicate to a user whether they were forking the discussion or simply replying to the end of the thread. Even when the tools did make this distinction, it seemed as if users were careless about where in the thread structure they added their entry, simply because it was not evident to them that this would be of any consequence.

2.4 Suggestions

Having examined a variety of existing discussion threads and forums, we found that there is some meta-data which we believe will useful in providing users with context.

We have compiled a list of these, focusing on those which are likely to help users find what they need, quickly and efficiently.

The list of meta-data or attributes we came up with can be divided into several different categories. The first category simply contains normal thread attributes, which are relatively mundane and easy to collect, but still essential. The second category contains meta-data not currently recorded or available in current discussion systems because they are unusual or challenging to collect. The third category contains attributes that are only relevant once users have already interacted with a thread in some way, by reading or contributing entries.

The first category includes obvious thread attributes such as subject, date of origin, date of last or most recent entry, total number of entries, author of originating entry, and author of last or most recent entry. It also contained entry attributes such as an entry's author, date, number of replies, parent entry, child entries, size, keywords, as well as how many users had viewed that entry already.

Some of the less obvious attributes fell under category two. These include aspects such as branching, bushiness, frequency, trends in authorship, etc. These will be described below in further detail.

One of the more unusual attributes, which we call "bushiness", refers to the amount of branching and sub-threads in a particular thread. A thread with a high degree of bushiness often has many entries posted in reply to any given entry, and thus has a high branching factor. This means that the "root" or originating entry might have many child entries, and each child entry might have many replies, and so forth. On the other

hand, a thread with little or no bushiness proceeds much like a calm conversation, where people always reply to the latest entry.

Frequency of posting or reading was another attribute which fell under the second category. Frequency refers to how many entries are posted within a given amount of time. It is obviously a relative quantity, since some threads might have hourly activity, while others might only receive postings several times a week. Nonetheless, it could be interesting to see when there were peaks in thread activity, and large amounts of contributions were made within a relatively short span of time.

Recency of posting or reading can be interesting because a thread's date of origin might not have anything to do with how recent the majority of its entries are. For example, a thread started a year ago might have postings which are very recent, whereas a thread started a week ago might only have postings from around the time of its origin. A thread with recent activity, or a thread with only old postings but a great deal of recent readership, might be more interesting to investigate. Many existing discussion interfaces only provide the date of thread origin or most recent posting which can be deceptive.

Length of contributions is also interesting because quite often users searching for some substantial amount of information can end up looking in the parts of threads where people are posting two word entries to each other (such as "I agree"), rather than at the parts of the thread containing original contributions to the topic under discussion. . If users were to know in advance the posting's length, they might be able to skip over entries which have only short remarks such as "I agree!" and "What do you mean?", allowing them to focus on the meaningful parts of the thread.

Trends in authorship and readership can reflect a great deal about what might be contained in a thread, the quality of the contents, and who is interested in it. It could be helpful to examine a thread and determine at a glance which and how many people had been writing or reading, and where. For example some threads have two people arguing back and forth over an issue and fifty people reading what they say. Other threads might have fifty or more people participating and only a couple non-contributing readers. These attributes might help users determine value of the topic to non-participants. Another use of these attributes could be to see where an expert in a field chooses to contribute a lot to a thread, or which threads an expert is choosing to read. Along the same lines, it would be nice to know how many people are contributing to a thread in total, as well as how many contributions in total any given author makes to a thread.

Keyword or semantic trends are something that we believe could be useful to users of discussion databases. A keyword trend is a semantic histogram calculated by examining entries in a thread for keywords, and then tracing whether or not there are any patterns in the sequence of entries. For example, a thread might start out with the keywords "domestic violence" and "abuse" appearing frequently in its early entries, but end with the keywords "alcoholism" and "counseling" occurring in its more recent entries. . Determining and showing this sort of trend could provide users with a much better idea of where the subject matter of discussions are going. Many threads have root entries with some particular subject, but end up on tangents far or different from their original purpose. This can be frustrating to users who delve into a thread, hoping to find some particular information, but end up wasting their time reading off-topic information.

Our third category of attributes is relevant only once a user has interacted with a thread, by reading or contributing to it. Users who have already shown interest in a thread might want to know if new postings have occurred since they last visited, and how many of these new postings have already been read by other users. They might also be interested to know how many people have read the same posting which they have read. More specifically, users may want to be aware of when a particular author posts something, or a particular user reads the thread. From our own experience, we feel it is useful to know when others reply our contributions, or when others reply to the same entry we have replied to.

In addition to the attributes listed above, we believe users could benefit from which threads are directed or problem oriented threads, and which threads are open-ended "opinion" threads, before starting to investigate them. In the case of directed threads, it would be especially helpful to indicate whether or not the problem posed in the thread has been solved. It may also be very useful to provide direct access to the postings that contain the problems and solutions.

2.5 Narrowing down the suggestions

With our list of attributes in hand, it was necessary to postulate as to which features which support them to incorporate into our design explorations. Some of these attributes were limited by social issues, some by implementation issues, and some by both.

For example, it was questionable whether or not reader tracking would actually be socially acceptable and technically feasible. This is because some individuals might not want others to know which entries or threads they had been reading, and would want to consider information of this nature confidential. It is also difficult to truly determine when a document has been read, not merely opened and then closed again, be it due to an accidental click or any other reason.

Keyword tracking is an example of an attribute which we decided not to focus on due to implementation issues. Effective keyword tracking is a technical thesis in itself, and is still a subject of on-going related research and speculation. It is definitely not something featured in current, publicly available databases, and beyond the scope of this thesis.

The attributes we chose to represent in our designs, as well as implement, were those which were more accessible to us and yet still potentially important to users. They will be described along with our proposed designs in the next chapter.

Chapter 3: The Design of the Visualization

3.1 Introduction to the Design

After investigation of several existing discussion databases, it was found that there were approximately three natural levels of detail needed in order to view and navigate a discussion forum.

The first level, which is the most distant or "macro" level view of a discussion database's information, allows the viewer to see all the categories of discussion threads available in the forum. For example, an introduction page to a web-based discussion database might present the user with a series of linked category names, such as Business/Finance, Civic/Government, Computers, Creative Arts, Current Events, Education, and so on. Users can click on any of these categories, and then be brought to another page or view.

Normally, at the first level, there is no information presented about individual discussion threads. In contrast, the second level provides some thread meta-data, although generally its main purpose is to display a list of thread titles. The meta-data available, such as the date of most recent activity, or the total number of entries in the thread, is displayed alongside the thread title. However, no thread structure and thread entry data is shown yet.

The level which follows level two is an overview of all the entries in the thread. Occasionally it displays thread structure, but typically it is a simple chronological listing of thread entry titles. From this third level, it is apparent which message is the root node

of the discussion, and which entries were posted after the root. Entry titles are often listed with their dates of submission and author names, and are sometimes accompanied by indications of whether or not a user has read them.

Clicking on an entry title in the third level brings the user to a fourth level of viewing, which is the most detailed or "micro" view possible in a discussion forum. Viewing the fourth level is similar to looking at an e-mail, except with discussion entry data instead. Usually the author, date, and title of the entry are displayed, along with the body of the text. In many cases, a "previous" button and a "next" button are provided for further thread navigation.

It was decided that our design would be divided into three main parts: Level One, Level Two, and Level Three. While our Level One and Level Two correspond to the general descriptions of the first and second levels above, our Level Three would display thread structure as well as entry content. We felt it would be effective to combine the levels three and four described above, in order to preserve the thread context of entries normally viewed in a separate level four. This would allow the visualization of thread structure to be used as a navigational tool for entry data, by providing direct access to the listing of entries at the same time as entry viewing.

The design work was divided among the team. Hyun-Yeul Lee worked primarily on Levels One and Three, while I worked on Level Two. Description of the entire design is necessary in order to convey the full picture of what we were trying to accomplish. However, since the bulk of my work was in Level Two, I will give only brief overviews of Levels One and Three below, and provide a more detailed description for Level Two.

3.2 Definitions and Design Principles

In order to design a visualization for discussion databases, it was necessary to experiment with certain basic elements of design, such as line thickness, color, macro and micro views, etc. There were other design concepts also important to consider and potentially integrate into our visualization. These involved techniques which are specific to multimedia environments, where animation and interactivity are available to add new dimensionality to a display.

As stated earlier, a large problem which occurs in current interfaces to threaded discussions is the lack of overall context provided to the user. Many of the principles which we kept in mind throughout our design are particularly relevant when presenting improved navigational context. We will describe these concepts in greater detail below.

Generally, transitions between any states in the display of data can be made smooth and gradual. While doing so, there can be a fine line between wasting the user's time and providing better context. Nonetheless, if these transitions are designed optimally, they help focus a user on whatever information is relevant to the transition, as well as provide that user with better context for the new screen state. When considered, sudden changes in the visual state of objects are a rare occurrence in a human's natural visual realm. Normally, the changes in the world we see around us tend to transition from one state the next in a somewhat gradual manner, as long as they hold our attention. The fact that we accept sudden screen changes as a normal aspect of computational display is not necessarily optimal. If chosen correctly, there are contexts in a multimedia representation of a data which could benefit from smooth on-screen state transitions.

These would be particularly useful for in the transitions between our levels, which we have designed to connect to one another smoothly, but in a new viewer's experience might connect less obviously. These levels are already categorized into macro and micro views of the data, and therefore it follows naturally that zooming [2] would be an animation technique we would consider. Increasing and decreasing levels of detail are always useful, and providing users with a transition will help users understand how one level corresponds to the next. Such a transition can achieve the effect of a zoom, even without being a "true" zoom into the picture, but it is important to give the viewer a feeling that the picture has just been magnified.

While designing our visualization, we also aimed for continuity in spatial layout. Evidently, users find it confusing when information they are accustomed to finding in a particular place on the page is moved to another area on the same page. To prevent this problem, we considered many strategies such as placeholders, animated transitions in layout, and continuity of general layout. For example, if a visualization of thread structure needed to shift substantially in order to accommodate new child entries, we would give additional consideration to redesigning the layout strategy of the thread structure, or design a better animated transition to the new layout.

It can be a challenge to focus the viewer's attention on a particular part of the screen, especially when navigating them through large amounts of data. The easiest and most brute force way is to display one thing at a time, showing only what you want the user to be looking at for that moment. This, however, succeeds in providing no navigational context. For example, a user looking at thread structure may want it available, even while browsing individual entry data. If both are kept on the screen at the

same time, the user is able to see where in the thread the entry being displayed comes from, and is not limited to only viewing the next entry in the thread. . This is much more efficient than having to switch back and forth between screens.

There are a multitude of subtle ways to shift a viewer's focus of attention, involving a variety of visual techniques and elements. One such way is by blurring elements not needing attention, and unblurring those which do. This can help soften background information without losing sight of it or letting it stray very far, since it could be interesting or useful one second later. It helps to alleviate the problem of having a cluttered page, without completely eliminating some of the information which should be kept visible but should not be distracting. Similarly, darkening, or reduction of saturation levels, in a background can help shift and focus attention. This results in the elements in focus becoming brighter and more attention-grabbing than those in the background. . It is also possible to use a fish-eye technique, where objects which are intended centers of attention are larger, and all the other objects appear incrementally smaller as they have less and less relation to the object in focus. The term "fish-eye" [3] is appropriate here because the technique is analogous to looking at something through a fish-eye lens: objects are scaled according to their proximity to the center of the picture. . Other elements in the environment are kept in view, but they are diminished, once again preserving context while minimizing distraction.

Shifts in focus can be needed to alert the viewer of some change in the state of the display. This is especially true when there are many small but similar elements on a screen, and a change in one small element might not be immediately apparent to the user. Techniques such as flashing objects or displaying large alerts are often employed in these

situations, but users dislike such disruptive notification, especially for a change of potential interest, and not of urgency. To accomplish a more subtle effect, we experimented with rippling the area of change on the screen to represent a change in the state of an element. In a rippling effect, the neighbors of the changed element are displaced slightly and then moved back, sequentially outward from the point of change, similar to the effect of a drop landing in water. This achieves an ambient and subtle way of bringing the user's attention to the area of change. At the same time, the ripple can easily be ignored if the viewer chooses to do so.

Another powerful visual technique is known as dynamic querying [4][5]. This allows a user to indicate, using a slider or some other control, a particular attribute's value which they are interested in, and watch as elements which have that value are highlighted. For example, if there was a slider bar which represented time, then a user could move the slider from beginning to end, and watch elements highlight according to their time-stamps. Similarly, a user viewing a thread visualization could click on someone in his or her buddy list, and see all entries authored by that person highlight. Using dynamic querying in combination with an effective visualization, users can find information according to different attribute values of interest, without having to face an entirely new screen of rearranged data.

The final but non-trivial technique which shall be mentioned here is known as the hover-over. A hover-over means that information pertaining to something on the screen is displayed when the user positions their mouse over a particular region of the screen. Usually this display occurs only after the mouse has entered the area and remained there for a certain amount of time (at least a second or two), otherwise the screen would flash

with unwanted information as the mouse is moved around. In current user interfaces, hover-overs are used occasionally, but most of the time this function is underutilized. Hover-overs allow non-committed browsing of additional, hidden, information and require minimal effort on the part of the user. We feel that they can be better utilized than they currently are, and this will become evident throughout the designs described later on in the chapter.

3.3 The Design Process

Finding visual metaphors to represent an online discussion database is an exploratory process. No "correct" answers exist when visualizing data, but there are representations which work better than others. Through our visual experiments, we arrived at the designs presented in this chapter. They are presented as possibilities, and not as end-all solutions. We went through many iterations of brainstorming, fine-tuning, and revisions to arrive at the designs proposed here, so it would not be feasible to include every version of each idea we experimented with here. However, some of the questions which were raised will be described below.

Discussion threads vary greatly in their structure, rate of growth, and activity in general, even if they are contained within the same discussion database. At the beginning of our explorations, we considered trying to break up the variability in thread structures and sizes, and design different visualizations for different threads. However, we decided that we wanted to focus on designing visualizations which would work for most common

threads. As a result, the possibilities we came up with are optimized for the middle ground.

However, at times we considered what to do in the case of data overflow. This is more evident in some areas of our design than others, since the problems we faced varied across the levels. Many times, it was necessary to consider numbers qualitatively, as our project was not focused on gathering the statistical data to support what "a lot" and "a little" really meant in the context of discussion data. Thus, in some of the designs below, one can get an idea of what kind of quantity is represented, but no real numbers are provided. In dealing with limited screen space, quantities are usually represented relative to one another, rather than as absolute values. We have represented this kind of information, so that viewers can get an idea of any quantity by viewing it comparatively with others.

We also had to deal with the issue regarding the type of information shown versus the amount of information shown. For example, there were a large number of attributes which we felt could be represented in our design, but there was always the danger of cluttering the screen or overwhelming the viewer with too much information packed into a small space. As a result, we chose to focus our design on only a few of the attributes which we felt were most important or most readily available.

Another trade-off we came across involved ease of use versus ease of learning. Some images and graphical user interfaces are extremely intuitive and easy to understand at a viewer's first glance. However, these might not be the most efficient to use later on, when time is a more pressing issue. On the other hand, some images or graphical user interfaces are extremely non-intuitive or difficult to learn for a first-time user, but are

easily and quickly utilized by an experienced user. For example, the pull-down menus in typical window-based software provide are easily found and provide clearly labeled access to many functions. Key commands for these same functions are often also available, but because they are less obvious to first time users, they are only used by experienced users, who might want to keep using the keyboard without stopping to reach for the mouse.

Throughout our designs, we aimed to maintain a particular aesthetic, in order to preserve the continuity between different levels of viewing. The mood of this aesthetic was first established during the design of Level One, and is kept in Levels Two and Three. Especially noticeable is the use of circular symbols. This choice was made to enhance the freshness and appeal of our designs, since much of on-screen design is rectangular. Our goal was to push the visual and interactive capabilities of our design, and thus we aimed to experiment with new, interesting, and atypical forms.

3.4 Brief Overview of Level One

Level One is has two main structures of information. The smaller structure is the field, which represents a subject area, for example "Pet Care". The larger structure is a grid of many fields. The general layout of Level One is shown in Figure 3.4.1.



Figure 3.4.1. Example Layout of Level One.

Each field contains one hundred tiny dots, representing individual threads, and each of these has several possible states. Threads can be active, or not active, depending on how recently activity has occurred in them, and this is represented by a bright red square with a yellow center, or a dark red square with an empty center, respectively. A yellow circle appears when a user has contributed to a thread, and appears as bright yellow when that contribution has been replied to. A blue circle shows that a user has read something in the thread, and appears as a brighter blue when new entries have been posted since the user last read. When a thread is hovered over, a little banner rolls out

from it, displaying the title of the thread. The symbols representing each state of a thread are outlined in Figure 3.4.2.

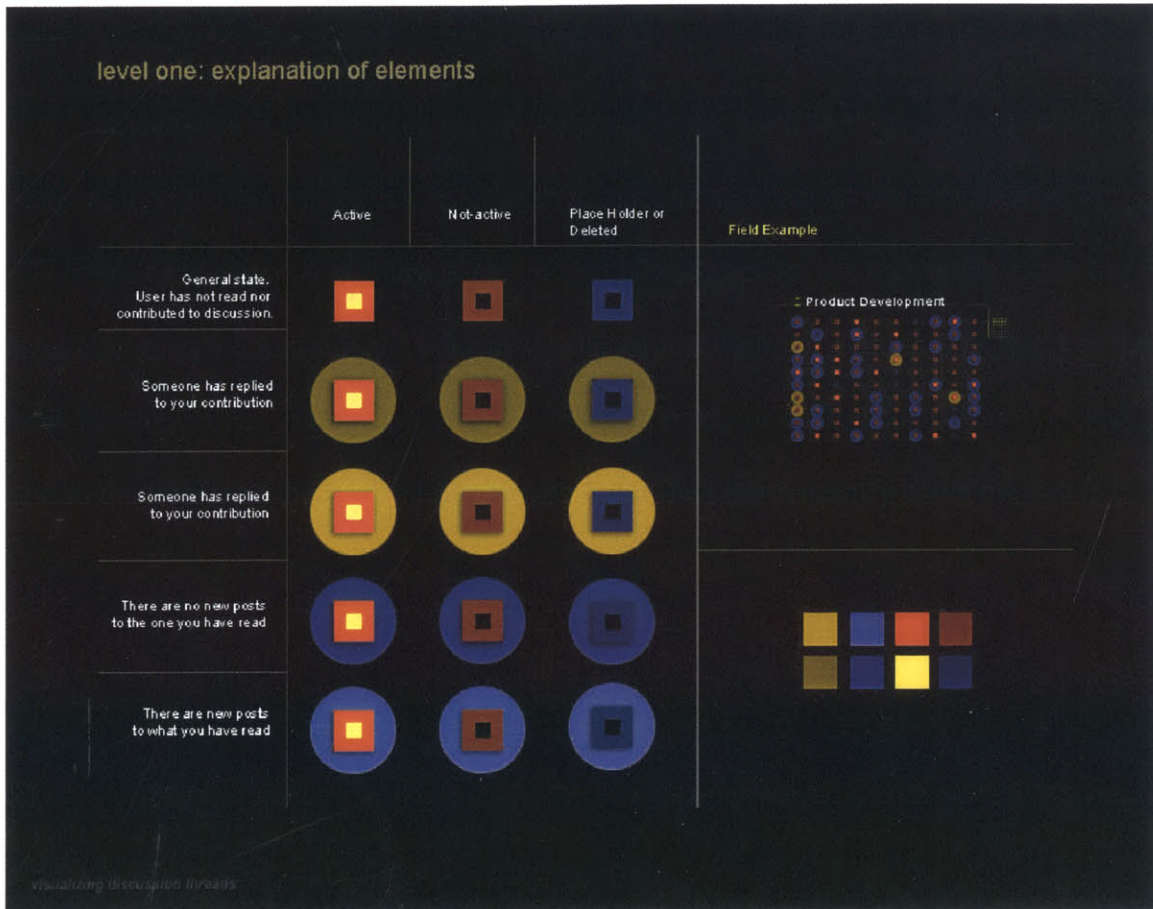


Figure 3.4.2. Different thread symbols for different states in Level One.

If there are more than one hundred threads in the field, then the field has several layers to it, and users can switch from layer to layer using controls which appear at the side of the field when several layers are present. To ensure that the user understands that these controls switch layers, the control has the same number of parts to it as layers in the field. In other words, if there are three layers in the field, three little controls will appear next to the field, and the one which corresponds to the layer currently being viewed is

highlighted. When the user clicks on another layer control, it highlights and the corresponding layer scrolls in. The user is shown the idea of layers because of the transition between them, and can always tell which layer they are on by looking at which part of the control is highlighted.

Level One is a grid containing many of these fields, three deep and indefinitely wide, each labeled with the title of its subject area. The fields are spaced far enough apart from each other that the user can still feel at ease faced with the large quantity of information, but within this constraint as many as possible are placed horizontally across the screen. Users can scroll left and right by throwing their mouse into the right or left margins of the screen, in which case the fields scroll right or left accordingly. This helps provide the user with a feeling of continuity in the layout of the entire level.

When the mouse is left within the area of a particular field for a certain amount of time, the field enters what can be called touring or Auto-Sweep mode. This is analogous to the sweeping that is done in radar systems, in which information is gathered and displayed about an area by repeatedly repositioning and sending out a signal. When a field goes into Auto-Sweep mode, the threads are highlighted one at a time, and the information that would normally not appear unless they are hovered over is displayed. This means that a user who wasn't sure which thread to enter in the field could just sit back and watch the titles scroll by for a while, until they saw one that caught their attention and decided to investigate it further. It saves a user having to drag the mouse over one thread at a time to see all their hover-overs.

3.5 Description of Level Two

Level Two corresponds to a type of viewing which is generally not used to display much information in existing interfaces. However, it has the potential to provide a great deal of navigational assistance to users browsing a database. This view is close enough to a thread to display appropriate meta-data, but far enough away to show it for several threads at a time. Thread meta-data can be extremely useful to the user; it can provide a better idea of which thread to investigate closely next, or even help determine if a particular thread is worth delving into at the time or not. This will become increasingly evident as further description of Level Two is provided.

The basic layout of Level Two corresponds to the layout of a field in Level One (see Figure 3.5.1). Like a Level One field, it contains many smaller controls, each representing a single thread, which are all laid out in a grid-like format. In this way, it can easily be seen as a "close-up" view of Level One, as if the user has just held a magnifying glass to a particular subject area field. Each thread is labeled with its title written above it. In order to help separate out one thread from the next and connect the thread representation with its corresponding title, and thin, light grey, stroke curves around the left side of each thread to the left side of its title.

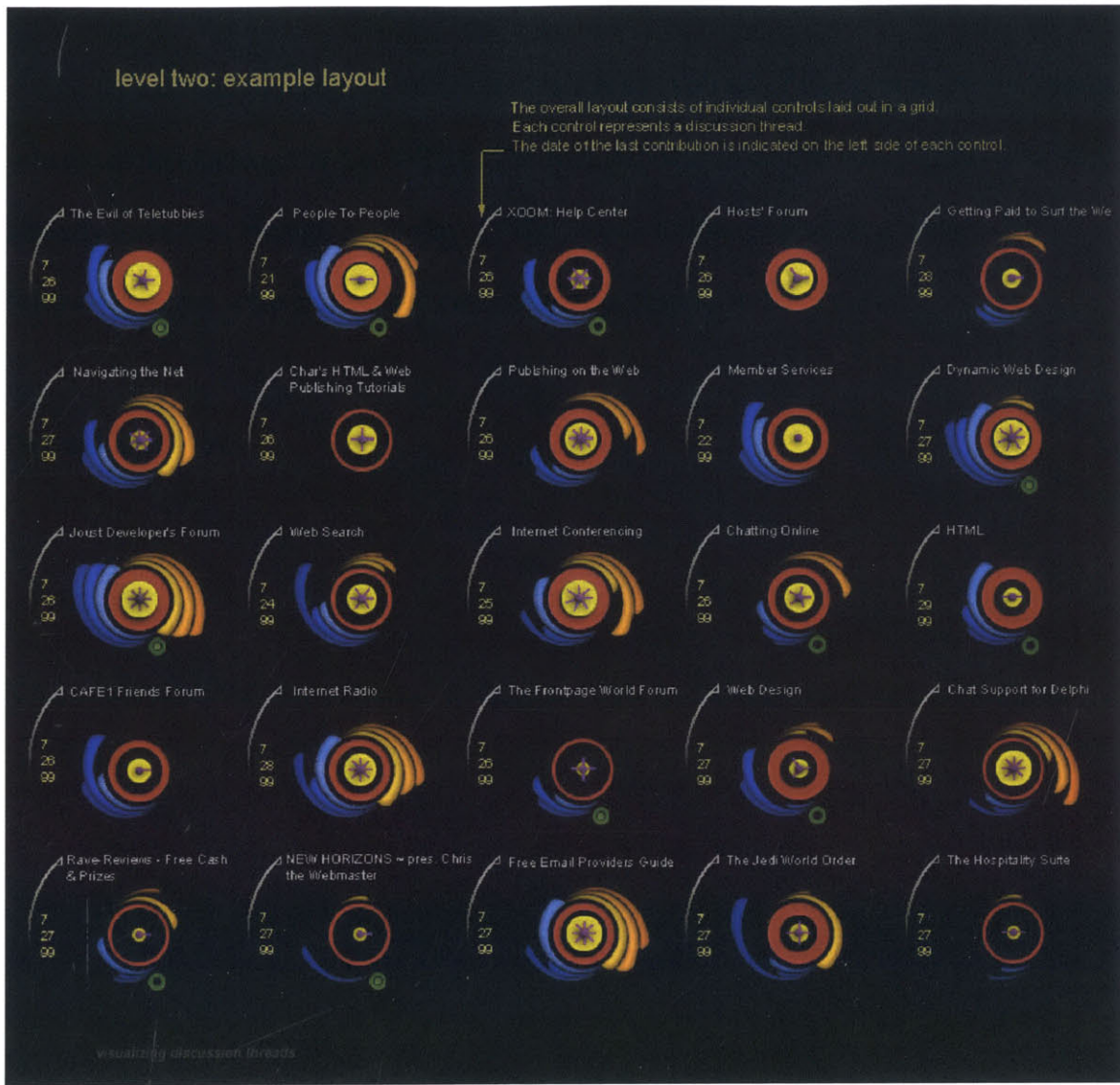


Figure 3.5.1 Example layout of Level Two.

The design of a control to represent a thread, with all of its appropriate meta-data, was the main challenge of Level Two. The first step was to decide which meta-data were actually important enough to display. Of the thread attributes described earlier, the following eleven were chosen and categorized for display in level two:

General attributes:

1. number of readers
2. number of authors
3. amount of business in the thread
4. problem thread or opinion thread
5. if problem thread, problem solved or not

Attributes pertaining to readership:

1. number of readers which have read as much as the user has read
2. number of new posts since the last time the user read
3. number of people that have read the new entries that were posted since the user last read

Attributes pertaining to authorship:

1. number of replies posted to the user's contribution
2. number of people who have read the user's contribution
3. number of people who replied to the same post that the user replied to

These were the data that we felt a user would be most interested in at the time. The same aesthetic and design could even be employed to display slightly different data attributes. However, since most of these attributes have never been provided at this level in most existing interfaces, we do not know for sure what the user response to them would be. Thus their inclusion is purely experimental, and can help us receive feedback from anyone viewing our demonstrations of the prototype. Note that the second two

categories of meta-data are relevant only if the user viewing Level Two has either read or contributed to a thread. This allows for the design to draw some simple parallels between the data in Level Two and the data in Level One, which also relies heavily on readership and authorship.

The fact that the control needed to represent and display so much different data at the same time presented an inherent design trade off between ease of use and ease of learning. It was decided after some debate that for our purposes, ease of use was of greater importance. Thus, the design is not necessarily intuitive to read for a first-time user, but a slightly more experienced user should be able to use the view to navigate extremely quickly. The experienced user would hopefully be able to determine which threads were probably worth visiting but just quickly glancing over Level Two. Thus the design concentrates on providing visual elements which change according to the data in a way which emphasizes them, in situations we believed would be more interesting to users. After much experimentation with the design, I arrived at the control shown in Figure 3.5.2.



Figure 3.5.2. Level Two control at full capacity.

It was apparent that a circular control would definitely be optimal. This corresponded well with the designs that were currently under way for Level One and

Level Three. The feathery elements around the outside were first inspired by the idea of a slider moving back and forth along a thin triangle, similarly to what is common in the temperature controls of a car. What is shown is the result of wrapping these thin triangles around the outside of the circle. Because the user will be faced with an entire field of these controls, it made more sense to lengthen and shorten the feathers according to the numbers they indicate, rather than have sliders along them. This results in a greater area of bright color around the threads which have a lot of activity, and therefore should need more attention, thus making them easier to pick out of the field at a glance. The colors of these feathery elements were chosen to correspond to those indicating readership and authorship in Level One, with the blue indicating that the user has read something in the thread, and the yellow indicating that they have contributed. However, most colors at this level were chosen at least partially because they matched with the scheme for Level One.

The center of the control is used to display most of the general information, such as total readership of the thread, which is indicated by the thickness of the red, and total number of authors in the thread, which is indicated by the thickness of the yellow. Threads with a large number of readers and a large number of contributors are made to stand out because they are bright red and yellow in their centers. The indication of thread bushiness is quite literal, showing many stems for a thread which has a lot of branching in its structure, and none for a thread which is one sequential line of replies. Finally, the green circle shows that the discussion was about a problem which needed solving, rather than about opinions. It is only present when the control represents a problem thread; otherwise that area is left intentionally blank. The color green can be associated with stoplights, and thus implies a certain amount of direction in the conversation. The green

sphere which appears in the middle of it shows that the problem has been solved. It serves to emphasize the green color which will be apparent to the user at a glance, and also completes the green circle which indicated a problem, in a kind of metaphorical completion of the topic.

The feather colors, although they are all blue for readership and all yellow for authorship, differ among themselves in brightness. Specifically, the feathers closest to the center of the control are the brightest, and the brightness decreases outward from the center, so that the outermost feathers are the darkest. The brightest feathers were chosen to represent what we felt was the most useful data, so that when they are full, the thread is brightly highlighted and calls attention to itself. Thus the brightest blue feather indicates how many new contributions have been posted since the user last read, and the brightest yellow feather indicates how many people have replied to the user's posting.

A legend for the different parts of the control and their various states is provided in Figure 3.5.3.

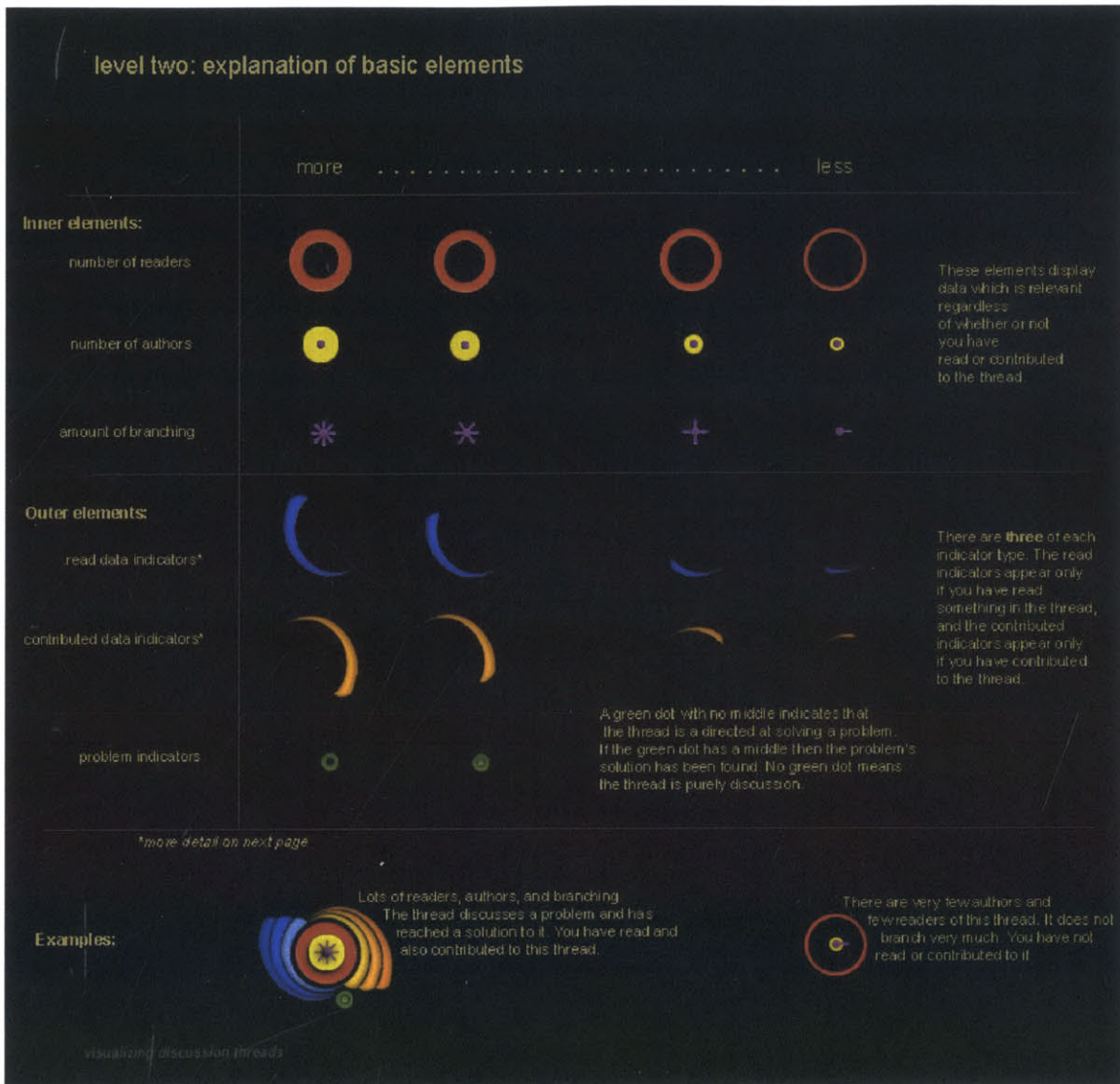


Figure 3.5.3. Legend for Level Two control parts and their possible states.

In order to help users learn how to read the Level Two thread representation, and also to provide them with a way to read real quantitative data about each thread, it was necessary to create a hover-over animation to label each aspect of the control. This too, presented quite a challenge, since normally hover-overs present a single element of data, not eleven pieces of data at once. The existing design inherently implied a spinning

time, all other threads in the field darken gradually. At the same time, the thread's title and side separator incrementally brighten, and the labels grow out from it. These two things help isolate the thread in question, by making it appear bright compared to the rest of the threads in the field. Labels for the side feathers follow the natural divisions between feathers, spinning out and then budding text bubbles. Readership and authorship numbers are labeled directly in the center of the control. If the thread is a problem, or a solved problem, the green spot, or the green spot and green ring respectively, are traced with a traveling light green pixel, and then connected to a horizontally growing label. The labels which indicated "problem" and indicate "solved" are placed very closely together, so that when they are both present they read like the phrase "problem solved". Each of the hover-over labels started out with the same color, but later on were changed to yellow, blue, and green, respectively, in order to better indicate which parts of the representation they were labeling.

The hover-over animation for Level Two provides labeling of eleven different data, and accomplishes in an extremely appealing way. In other words, even though it is not necessarily an extremely efficient way of labeling a visual representation, it has an aspect of memorability to it, which will help users remember the labels the next time they see it, and thus be able to read the control more and more quickly every time. However, there are certain border conditions for which the hover-over animation so far will not be effective: specifically, the edges and corners of the field. In order to solve this problem, the labels for the side experiencing overflow could simply overshoot their normal ending positions, and instead spin around the control until they are away from the edge of the page. This is illustrated in Figure 3.5.5.

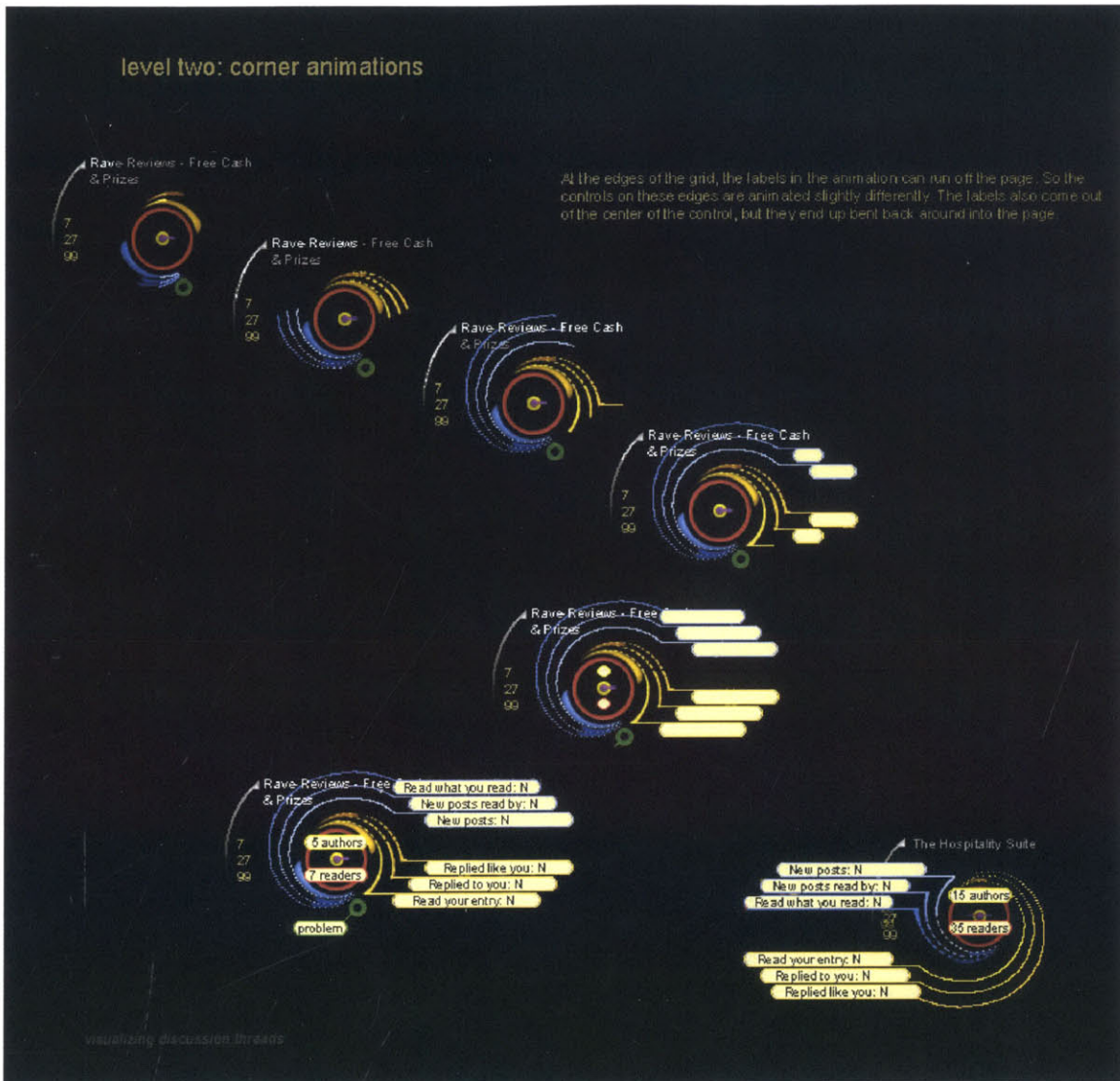


Figure 3.5.5. Corner condition hover-animation for a Level Two control.

The final effect of Level Two is that threads with greater activity, whether it is general activity, or activity pertinent only to the current user, should stand out more in the field.

3.6 Brief Overview of Level Three

Level Three provides a visualization of overall thread structure, as well as ways to view individual entry data and navigate entry data using the thread structure as a map. It consists primarily of connected circles, where each circle represents an entry in the thread, and connectors link entries with their replies. The primary design challenges faced at this level included how to grow the thread optimally (i.e. in order to avoid entire rearrangement of the space), and how to represent different contributors in the thread. The design was never truly finalized, but the basic ideas behind how it operates will be described here.

The thread structure starts out with a single root node, which has a special color. Replies to an entry are placed around their parent in a circle, sitting in chronological order, clockwise, with the first reply at three o'clock. When there are more replies to an entry than fit with the current radius, an animation is used to show the user how the entries will be rearranged. The radius at which the child entries sit is increased, and the entire existing structure is swung around counterclockwise as the child entries are recompressed backwards around the circle. If there is spatial conflict between entries, in other words, the rules for display mean that two or more things need to be in the same place, those entries on a branch added layer are moved to a lower layer. This means that they appear to be underneath the earlier data, but can be brought forward by placing the mouse over them.

Each entry is represented by a little circle, but there are many different kinds of little circles, and each one represents a different user's contributions. Also, a list of people

can be dynamically queried to show all entries posted by a particular user. Entries can be clicked on to view the entry data as well. The basic layout is illustrated in the Figure 3.6.1.

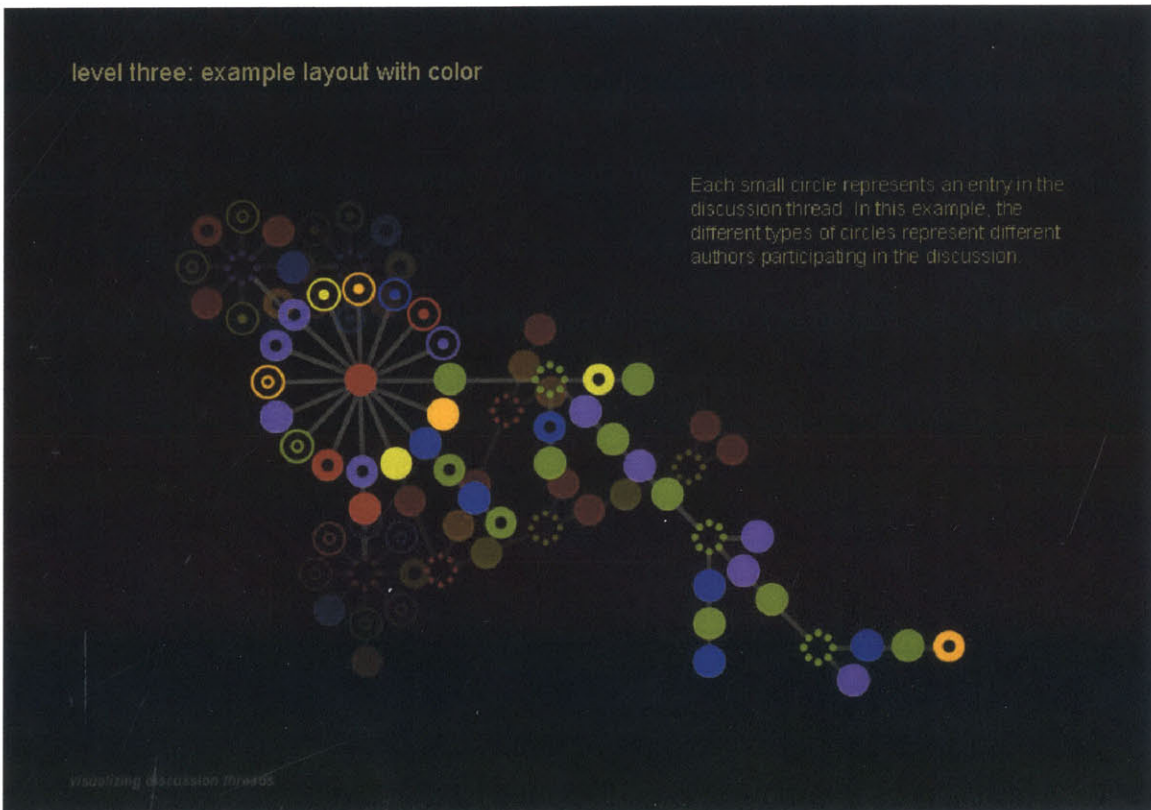


Figure 3.6.1. Example Layout of Level Three.

Level Three was also designed with capabilities for dynamic query, by various entry attribute values. An example of this dynamic query by author name is shown in Figure 3.6.2. Two names are selected in a buddy list, where they are shown with their special symbols. Correspondingly, the nodes representing entries by these selected authors are highlighted in color within the thread structure. On the left side of the thread

structure is an example of how entry content might be displayed, when a user clicks on a particular node in the thread.

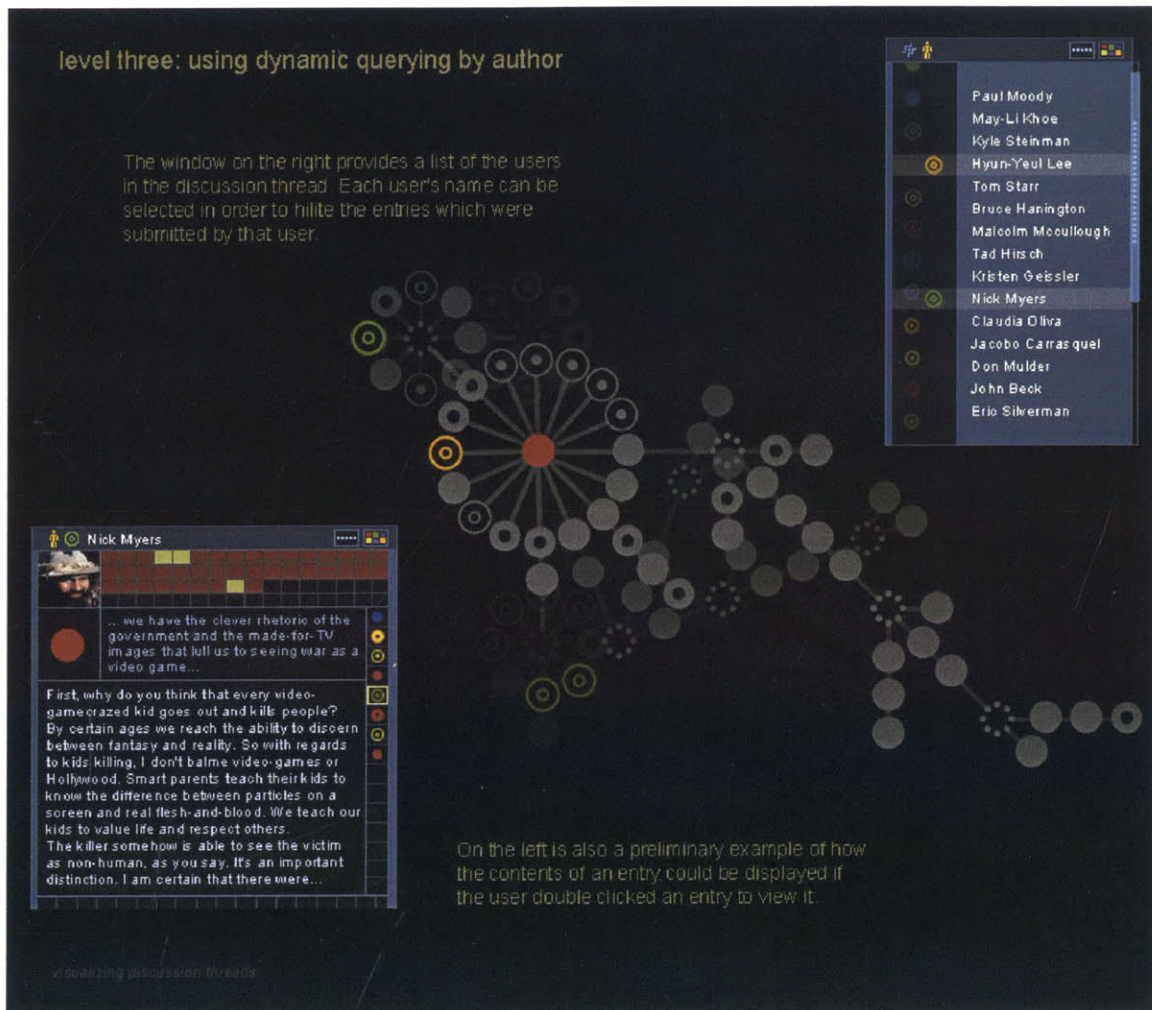


Figure 3.6.2. Level Three layout with a Dynamic Query by author name.

Chapter 4: Experimenting with Implementation

4.1 Introduction to the Prototype

Some consideration was required in choosing which parts of the design to implement in a prototype so we could experience it firsthand. It would not be possible to create the entire design, given the scope of this thesis and the time constraints we were facing. The entire design would have required creating and customizing a new server back-end to host discussion databases, and the collection of the new meta-data that our designs required.

The most useful, relevant, and feasible part of our design was level three, the visualization that displayed a single thread's entire structure while displaying one of the thread's entries. The types of meta-data required to implement level two would not be readily available in current discussion database systems. The type of meta-data needed to implement level one would also be unavailable, namely information pertaining to readership and authorship for a given user. Thus level three offered an interesting visualization of a thread's structure, and the most feasibility given the time and scope of this thesis.

Before continuing on into greater description of our prototype, it would be advantageous to explain some of the Director terminology we will be using. A Director "cast" corresponds to the cast of characters in a play, and contains all the possible elements which could appear on the screen, referred to as the stage. A Director "sprite" is a cast member which is on the stage. The "score" in Director works like the score for an

orchestra: it provides an axis representing time, and shows which sprites are on the stage, when, and for how long.

The implementation of our prototype was an experimental combination of several different tools and environments. This combination included Macromedia Director, XML, and Lotus Notes, although some initial experimentation included Javascript as well. The reasoning behind each choice of tool or environment, as well as the manner in which it was used and results of its use, are discussed below.

4.2 Director as a Front End to Notes

Macromedia Director was chosen as the experimental medium in which to create the graphical user interface for a variety of reasons. Director has built-in multimedia capabilities which can easily be packaged for the web, via the ubiquitous Shockwave format. Director supports the use of graphics and animation, in conjunction with other visual techniques which help support context, such as fades, shadows, and background transparencies. These techniques can be hard to achieve smoothly and effectively in other media, especially when they are displayed over a web interface. It was apparent at the time that some of the manipulations which we intended to do in our visual representations could be efficiently done using Director given its ability to easily handle animation.

Because of Director's multimedia capabilities and its accessibility via the web, Director is already widely used by high-end web designers. This is an added incentive for a corporate research project, since it would obviously be beneficial for a company's back-end products to be easily combined with high-end multimedia web design.

Director also showed a lot of potential for programmability using its object-oriented scripting language called Lingo. This allows for various visual attributes of objects to be manipulated programmatically, using scripts which can be triggered by user input. Lingo also features a data structure known as a propertyList, which can be used to create tables and other in-memory entities that facilitate efficient storage, manipulation and retrieval of large amounts of structured data. Changes in data which occur at the back end can also be polled periodically using a timer, although "pushing" data from Notes to the web would not be implemented so easily. With the existence of Director Xtras, programmability increases even further. Director Xtras are extensions to Director and allow it to perform particular functions which are computationally more complex than regular Lingo scripting allows for. Specifically, Director comes with an XML Parser Xtra, which showed great potential for our purposes.

Another aspect of using Director which is advantageous is that the resulting shockwave files are browser independent on the web. While many possible methods for rendering and manipulating highly visual content for the web exist, quite often it is complicated to keep track of the possibilities and standards which are provided in the two largest competing browsers, Microsoft Internet Explorer and Netscape Navigator. As a result, one quite often has to create two different versions of a prototype in order to achieve the same visual effects, and then call them depending on which browser is detected to be in use. Shockwave files, on the other hand, although they require the installation of a plug-in in order to view them, are viewed identically on both browser types.

Lotus Notes was chosen as the back-end database to host and store the discussion databases we would be interpreting and displaying via Director. The reason for this choice was largely due to time constraints as well as the large amount of expertise in using Notes which is, quite naturally, available at Lotus. Also, Notes development is starting to provide more web features which take advantage of XML, something which showed potential for our project. Notes provides a standard discussion database which is can be manipulated using the Notes development environment. The discussion database provides for some of the data structure that we are interested in visualizing, such as parent-child relationships, in the entry data itself. This means that rather than parsing ordered list indentations manually from HTML in order to extrapolate these relationships, they could be directly accessed instead. Thus, rather than build a back-end discussion database from scratch with all our desired functionality, or even minimum functionality, or use one which was unfamiliar to anyone in the group, we chose to use Notes.

A potential drawback to using Director's programming capabilities, as well as using Notes, was the issue of speed. The performance of the Director scripts in combination with web access of Notes data could potentially be unacceptably slow for our purposes, or at least for any expansion of our purposes. Perhaps we could obtain a satisfactory performance for the prototype, but not for any real application of the interface. However, this would only be evident after experimenting with some substantial amount of data using the prototype. We considered using Java because of its new 2D API, which would allow a similar manipulation of images, but Java is not known for its speed either. So it was decided that Director would be our choice of medium, provided we could determine that it could communicate with the necessary components of the

system, as well as provide us with enough functionality to perform the required tasks. Director would need to communicate with Notes over the network somehow, and obtain all the back-end data necessary to render the appropriate visual representation. Director would need to be able to create sprites dynamically to create new visual objects on the fly that directly represent some of the data retrieved from the back end. This might seem like an intuitive and obviously necessary function to include in Director's capabilities, but there was some doubt as to whether or not it was possible to execute in an elegant manner. These matters will be discussed further in the next three sections.

4.3 Network Communication between Notes and Director

The first exploration was to determine how Director and Notes were going to communicate over the network. There were two methods that were experimented with: the first was an attempt using Javascript as a middle ground between Notes and Director, and the second involved using network commands directly from Director to Notes. The first method will be described briefly here, as it may be of interest to those who wish to use Javascript-Director communication. Following that, the second method shall be described in better detail.

The experiment using Javascript consisted of four parts: communicating from Director to Javascript, Javascript to Director, Notes to Javascript, and Javascript to Notes. Our investigation ended after the first two parts were completed, since we were able to determine a better method for direct communication between Director and Notes instead.

The communication between Javascript and Director operates as a push from either side. If the Javascript side wants to send information to Director, it can call a method called EvalScript(), which takes a string, and acts on the director object embedded in the same page. For example, if a director file is named test.dcr, then within the HTML page there would be some HTML saying:

```
<EMBED SRC="test.dcr" NAME="test" SWLIVECONNECT=TRUE WIDTH=....etc>
```

Within the Javascript on that page, one could define a function which calls Director using the method as follows: `test.EvalScript("teststring")`. In order to receive the event generated by Javascript, the Director file also needs to have a corresponding method defined, which takes in the string sent by the Javascript. In the other direction, Director can also call functions in and pass strings to the Javascript. To accomplish this, nothing special has to exist within the Javascript other than the desired function definition.

Using Javascript was eventually abandoned for several reasons. It reintroduces browser dependency, since the communication did not work the same way in Internet Explorer and in Netscape. Also, because it was a two-part transaction in either direction between Notes and Director, it looked like it would be relatively slow and inefficient compared to other possible methods. Lastly, realizing that Director had commands to import the source of web pages, specified by URL over the network, we saw this as a better method. .

Our final method took advantage of a two seemingly inconspicuous features in Director and Notes which had never been used together before. One feature, which has been alluded to already, allows Director to parse the HTML source of any page linked to the web, simply by using the command `getNetText`, and specifying the URL of the page to grab. Of course, there is some delay over the network, and the Director file has to be

scripted with functions which check on the status of its network request, but the basic functionality exists. Correspondingly, there is a feature which exists in Notes and Domino known as the "URL command". Several of different kinds of these URL commands exist; they allow access to servers, databases, and other components of a Domino web site, all by creating a link or entering a command into a browser. As a result, databases, views, and documents in the back-end discussion database can be accessed using a URL command. In combination with the Director command, `getNetText`, these URL commands allow for Director to directly access the Notes/Domino back end discussion database.

The `getNetText` function retrieves the source of a given URL, and URL commands allow retrieval of specific back-end data given a specified URL. These two features seem to fit so well together, that they appear to be what is needed in order to achieve direct Director-Notes communication. It is also noteworthy that the `getNetText` function works in conjunction with related functions, which allow error codes and retrieval status to be determined at any time. This is important, since in order for the system to be even slightly robust, error detection and isolation when dealing with network transactions is not optional.

4.4 Managing the Data Communicated between Notes and Director

Although the mechanisms described in section 4.2 solve the problem of communicating data between Notes and Director over a network, there still remains the question of what form the data should be communicated in. If the data is retrieved from

Notes by Director in a form which provides no meaning for each entry, for example, a simple HTML ordered list structure, there will be a lot of computation needed on the Director side. This additional work would be needed in order to determine contextual data, such as the parent-child relationships between discussion entries.

Since the Notes back-end naturally provides this sort of specialized data within the Notes development environment, it would make sense that Notes be able to provide this sort of information to outside sources as well. To use a simple URL command in order to retrieve a Notes view would result in Director receiving the HTML source of a page which displays a notes view. However, since HTML only provides browsers with information on how data should be displayed, all the information which Notes normally holds within a view would be lost.

This is where XML can provide us with some helpful structural data. XML, also known as the Extensible Markup Language, allows tagging of textual data according to its relevance and meaning, rather than simply according to how it ought to be displayed by a browser. Using plain HTML, a discussion entry's date would simply be retrieved by Director as possibly some bolded text, i.e. `December 6, 1999`, in which case the front-end would have to determine, simply by analyzing format and perhaps matching strings, its relevance as the date of an entry. This could be prone to error and require a lot of extra computation, since view structure can vary and a lot of string manipulations and counters would be required. However, using XML, this same data can actually be tagged as being the date of an entry using a specified XML tag, i.e. `<date>December 6, 1999</date>`.

Lotus Notes actually provides a URL command which generates XML representing Notes data. This XML functionality provided by Notes is still in its preliminary stages, and has not yet been completely standardized. However, for our purposes, which are largely experimental, we felt that the little that was provided at the time was at an advanced enough stage to try using it. The URL command `ReadViewEntries` allows one to retrieve the XML representing the data by specifying an absolute starting index of an entry in a particular view, and a count to indicate the number of entries at and after the starting index to retrieve. By absolute, I mean that each entry is named one after the other, regardless of their parent-child relationships within the thread. For example, if entry number 4 has three children, and entries 1 through 4 are requested for retrieval, the 3 children of entry number 4 are not retrieved, since they are considered entries 5, 6 and 7. The ability to retrieve XML formatted text helps immensely in retrieving the data to display the overall layout of a thread, i.e. level three, since the default view of a discussion database provided contains the titles, dates, and some structural data of all the entries in a thread. However, there are a few technicalities which accompany using the `ReadViewEntries` URL command, and a few inconsistencies. Some of these are due to the preliminary nature of the Notes XML features, and some are due to the fact that the Notes development environment and its data structures don't necessarily translate elegantly when used by external environments.

For example, despite the fact that `ReadViewEntries` retrieves the entries in a view using indices which count entry by entry, regardless of the parent child relationships, no provision is supplied in order to determine in advance what the total number of entries in the view is. There is only some data at the beginning of the retrieved XML view data

which states how many top level entries there are in the view. It looks something as follows: `<viewentries toplevelentries="1">`. This means that if a thread has been started using one entry, and all other entries are descendents of this initial entry, then using a URL command one would only be able to determine the existence of one top level entry. For our purposes, this data is not sufficient. Therefore, in order to determine how many entries there are in total, a flattened view was created within the back-end in Notes. This flattened view, named FlatView, contains all the same entries as the regular default view, but it considers all entries to be top-level entries. Thus when it is retrieved using the ReadViewEntries URL command, it actually provides Director with a number which is a true count of the total number of view entries. Notice that the retrieval process now ends up being a two-step procedure. The first is to retrieve just the 1st entry of the flattened view, using the URL command which looks like the following:

```
http://server/databasename.nsf/FlatView?ReadViewEntries&Start=1&Count=1
```

This would be enough for Director to obtain the XML which contains the total number of entries in the view, namely: `<viewentries toplevelentries="<n">`, where n is the integer value of the total entries. After this, Director can make a second request to the back-end, which would look something like:

```
http://server/databasename.nsf/MainView?ReadViewEntries&Start=1&Count=n
```

and this would cause Notes to return all the rest of the data in the view to Director.

Also noteworthy at this point is that, according to our design of level three, we would ideally like to be able to display author names and perhaps even the textual body of any given discussion entry, at a click of the mouse. However, the only existing URL command which generates XML at the time could only do so for the data provided within a view, and could not retrieve document data. At the same time, the Notes default view

for discussion databases does not always provide an entry's author's name, and it definitely never provides the actual content of the entry's body within the view. Thus, a specialized view was created to this effect, containing the body of each entry within the view. As a result, the second request which Director makes to Notes results in the retrieval of all necessary discussion data, including the body of each entry itself. Here is a sample of how the XML data looks when it arrives at the Director front end:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Lotus-Domino (Build V502_09141999 - September 14, 1999 on Windows
NT/Intel) -->
<viewentries toplevelentries="2">

<viewentry position="1" unid="E37D7655600794A68525683F0068C2BC"
noteid="8FA" children="2"
descendants="2" siblings="1">
<entrydata columnnumber="0" name="$106">
<datetime>19991206T140415,96+05</datetime></entrydata>
<entrydata columnnumber="1" name="$116">
<text>2</text></entrydata>
<entrydata columnnumber="2" name="$120">
<text>hello? is anyone there? (May-Li Khoe)</text></entrydata>
<entrydata columnnumber="4" name="Body">
<text>Is anyone else posting messages in this newsgroup? Anyone out
there?</text></entrydata>
</viewentry>
```

So far Director has succeeded in retrieving the discussion data in XML format from the Notes back-end. At this point, Director's XML Xtra could be used to help parse the necessary data from within the XML. The XML Xtra in Director allows XML data to be indexed and parsed easily. Thus, rather than having to go through the data manually, using numerous counters and string comparisons, we could call the Xtra's methods on the imported discussion data in order to retrieve the entry data by index. Because the XML methods such as `getAttributeName` and `getAttributeValue`, listed in the help documents, were not fully functional at the time, it was necessary to access data by index number only. Because of this we analyzed the XML data in order to determine which index corresponded to what desired values. For a particular field, the index needed could be

different from entry to entry, depending on what relationship that entry had to others. However, there was definitely a pattern to the numbering schema, which was determined and then put into practice through a series of if-then statements.

4.5 Managing the Data within Director

All this data could be retrieved from the XML using Director's XML Parser Xtra, but the data still needed to be stored within Director's memory. This was accomplished using a data type in Director's Lingo called a propertyList.

Lingo propertyLists allow storage of a list of name-value pairs, which are also retrievable using various different methods. For example, the values can be accessed using name or index number (i.e. position in the list). It is also possible to obtain a pair's position number or index in the list using the value. Most importantly, propertyLists can be contained within one another, which allows for a table structure to be created.

The Notes XML data which is received by Director arrives with entries listed in a depth first manner. This means that if a particular entry has three child entries, it is listed first, followed by its three children in chronological order. This is true for each entry in the list. Every entry also has a position number, which is assigned to it within a Notes view, and then sent to Director along with the rest of the XML data. These position numbers reveal the line of descendants belonging to each node. For example, a root level node might simply have position number 1. The children of this node at position 1 will have position numbers 1.1, 1.2, 1.3, etc., accordingly. If the entry with position number

1.2 has two children, they will have position numbers 1.2.1 and 1.2.2, respectively, and so on.

In order to store each entry's data in a manner easily completed from the XML data provided, but still easily navigable from parent to child and back, a special data structure was necessary. This data structure consists of three levels of propertyLists: the top level propertyList simply contains all the entries, the second level propertyLists contains each entry's data, and the third level of propertyLists are list of an entry's children. Each entry is stored with its position number as its name in the main, top level propertyList. As its value, each entry has a sub-propertyList, containing the main data fields, such as "title", "body", "parent", and "kids". The value of the last field, "kids", is also a sub-propertyList containing all the position numbers of the child entries. Thus, in order to trace from an entry to all of its child entries, one simply has to iterate through the propertyList in its "kids" field, and use the position numbers in that list to access the top-level propertyList accordingly. Conversely, to reach a parent from a child, one simply has to access the top-level propertyList with the value of the entry's "parent" field.

This structure is illustrated by Figure 4.5.1.

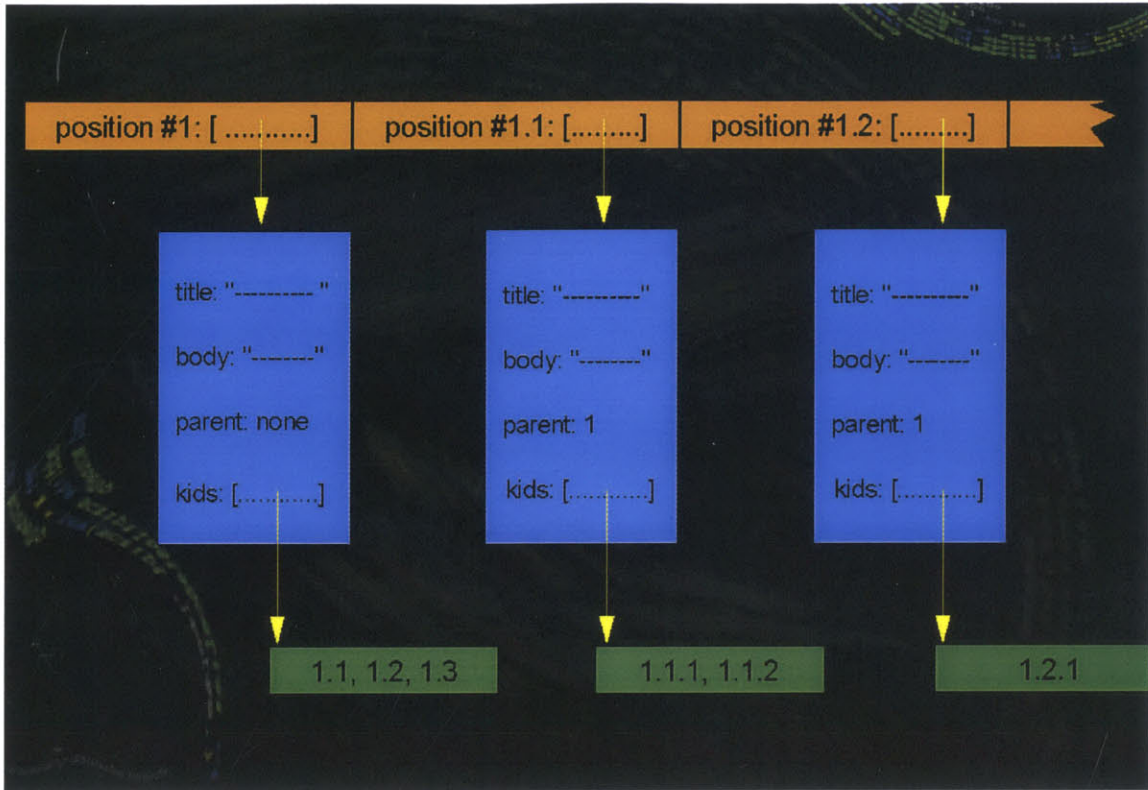


Figure 4.5.1. Storage Structure of example discussion data.

Using this basic structure, the data can easily be accessed for display to the screen. Also, additional data, which might be needed to keep track of an element's desired position on the screen or other visual attributes, can be added to the second-level data propertyList. Although access speed required to manipulate large lists would normally be a consideration, there is no other data type or structure within Lingo to perform a similar function. Thus for our purposes this is sufficient.

4.6 Creating Sprites Dynamically

The visualization of the discussion database consists of a number of elements, which need to react to the user's mouse events, and then display information according to which entry they represent. At the same time, there is no way of knowing how many entries will be present in the database before that information is retrieved over the network. As a result, the visual elements of Director, called sprites, needed to be created on the fly. Unfortunately, Director was designed more as a presentation-type tool, in which all the sprites are known of and created manually in advance. Thus some dynamic manipulation of Director's current functionality was needed in order to achieve the desired effects.

One possible implementation was to attempt creation of the maximum number of sprites possible, and hide those intended to represent and connect the entries off screen, only making them visible when needed. However, predetermining the number of sprites needed could fail eventually, since if there is even one document extra, it would be impossible to create that extra sprite at runtime, and then representation would be inaccurate. Thus this method did not appear to be a robust solution

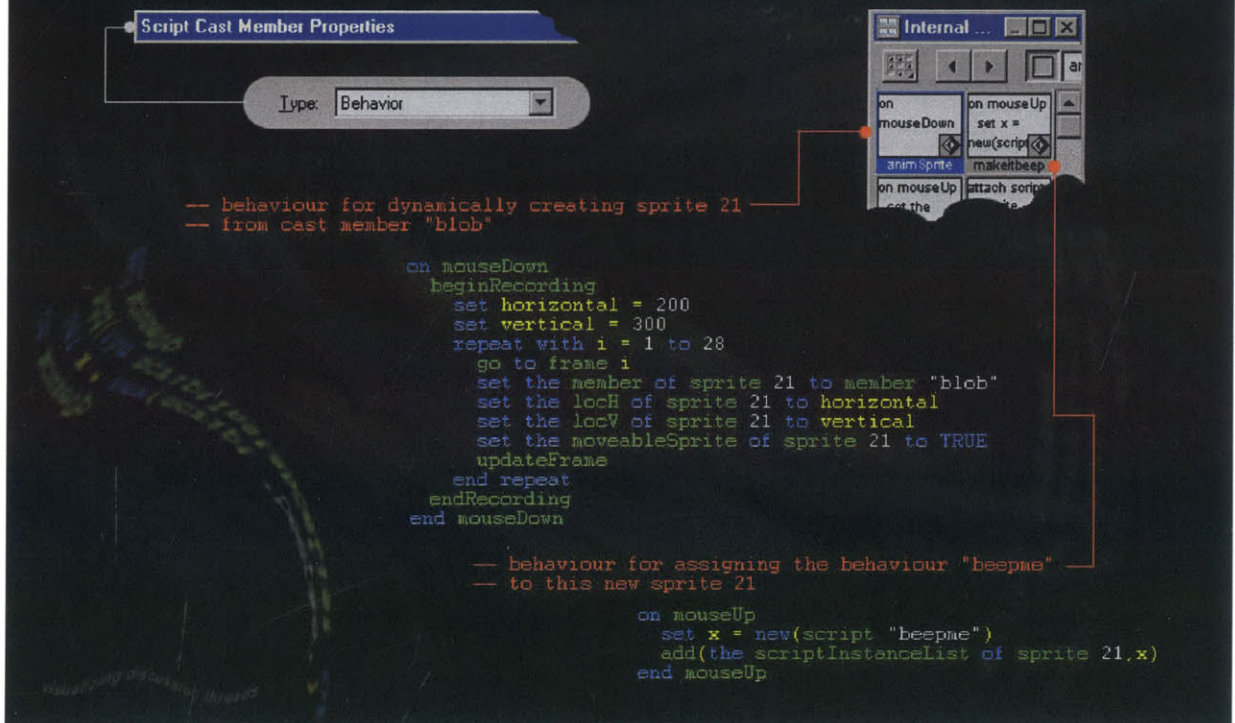
It was apparent that Director was not yet intended to be a medium that was altered at runtime. However, there was one seemingly obscure way to dynamically create sprites using Lingo, and this was to take advantage of the commands `beginRecording` and `endRecording`. This method seemed slightly more elegant, in that it allowed for new sprites to be added into the score to correspond with each entry in the discussion database. The documentation we created to describe this technique is shown in Figure 4.7.1.

investigating director: dynamically creating sprites

In order for a sprite in Director to represent an interactive object in a Notes View, the sprite must be dynamically created and assigned behaviours. This is because the Notes view objects can change constantly, and thus a representation of them must also be dynamic.

Director, however, was designed for the manual creation of sprites at the time of movie creation, via its graphical user interface; and not at movie run time, via script. The latter option is not explicitly supported, and it was necessary to do some exploration of Director's Lingo scripting potential in order to discover an effective but undocumented approach.

In order to accomplish the dynamic creation of sprites, we have defined behaviours using the scripts below. This code shows how to first create a moveable sprite and then assign a behaviour to it. The two pieces of code can be combined if necessary.



The screenshot shows the Director interface. On the left, the 'Script Cast Member Properties' window is open, with the 'Type' dropdown set to 'Behavior'. On the right, an 'Internal ...' window displays Lingo code. The code is divided into two sections by comments. The first section, 'behaviour for dynamically creating sprite 21 from cast member "blob"', contains an 'on mouseDown' handler that records the mouse position, repeats a loop to go to frame 1, sets the location and moveable properties of sprite 21, and updates the frame. The second section, 'behaviour for assigning the behaviour "beepme" to this new sprite 21', contains an 'on mouseUp' handler that creates a new 'beepme' script instance and adds it to the sprite's script instance list.

```
-- behaviour for dynamically creating sprite 21
-- from cast member "blob"

on mouseDown
  beginRecording
  set horizontal = 200
  set vertical = 300
  repeat with i = 1 to 28
    go to frame i
    set the member of sprite 21 to member "blob"
    set the locH of sprite 21 to horizontal
    set the locV of sprite 21 to vertical
    set the moveableSprite of sprite 21 to TRUE
  updateFrame
  end repeat
endRecording
end mouseDown

-- behaviour for assigning the behaviour "beepme"
-- to this new sprite 21

on mouseUp
  set x = new(script "beepme")
  add(the scriptInstanceList of sprite 21,x)
end mouseUp
```

Figure 4.7.1. Documentation showing a basic example of how to create sprites on the fly and assign behavior to them dynamically.

However, it also had drawbacks, in that it could only add sprites to one frame of the movie at a time, and this caused the screen to flash. It also sometimes caused a strange corruption in the transparency of sprites that were on screen at the time of the recording. In the end, the movie was created to be only a few frames long which looped,

and the flashing could be reduced to one single flash. The movie was also partitioned into two sections, one section which allowed the retrieval and parsing of data, and the other which allowed the display of the elements. These could be thought of as the two states of the movie: data retrieval and data display. As a result, the transparency side effects which could occur during the recording time would be isolated within the first section of the movie, but since the recording happened only in second section, those side effects were never seen by the user.

Once the sprites were on the screen, they also needed to react to user input according to which discussion entry they represented. This meant that behavior scripts had to be assigned to them on the fly as well - another task utilizing an obscure part of Director's functionality. This was accomplished using something called the `scriptInstanceList`, which keeps track of which scripts a sprites uses. The sprites which were dynamically created each had to have sprite numbers (this corresponds to their placement in the score). Adding this sprite number to a script's `scriptInstanceList` meant that the corresponding sprite would acquire that behavior.

4.7 Overview of the Implementation

In order to provide the reader with a better idea of how all these parts collaborate, a brief overview of the prototype as a whole will be described here.

At run-time, the prototype basically executes the following steps:

1. retrieve the total number of discussion entries from FlatView in Notes
2. retrieve the actual entry data using Notes URL command `ReadViewEntries`

3. parse and create the data structure containing all entry data
4. create new sprites and assign behaviors to them according to the data

Note that for steps one and two, the prototype enters a loop in which it checks the status of its network request. If it receives an error, it notifies the user of the nature of the error, which is determinable using the error code provided by a command in Director. If it receives no error and the network transaction is complete, i.e. `getNetStatus` returns 1, then it no longer checks network status and continues on to step three to parse the data.

Figure 4.7.1 shows the Lotus Notes discussion data which is being retrieved by the prototype in the screenshots below, as it would normally be displayed by the Notes client.

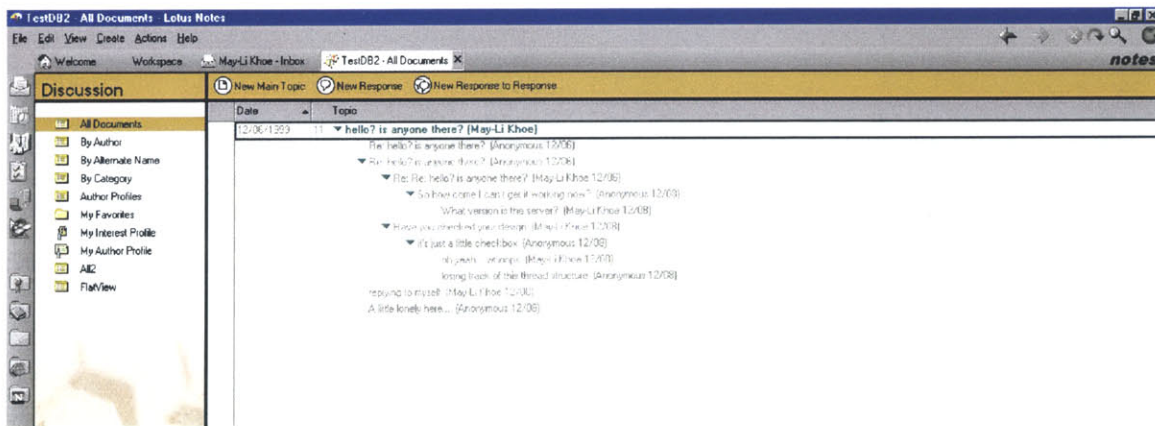


Figure 4.7.1. A standard Lotus Notes view of a discussion database.

Screenshots of the prototype are provided below in Figures 4.7.2 through 4.7.5.

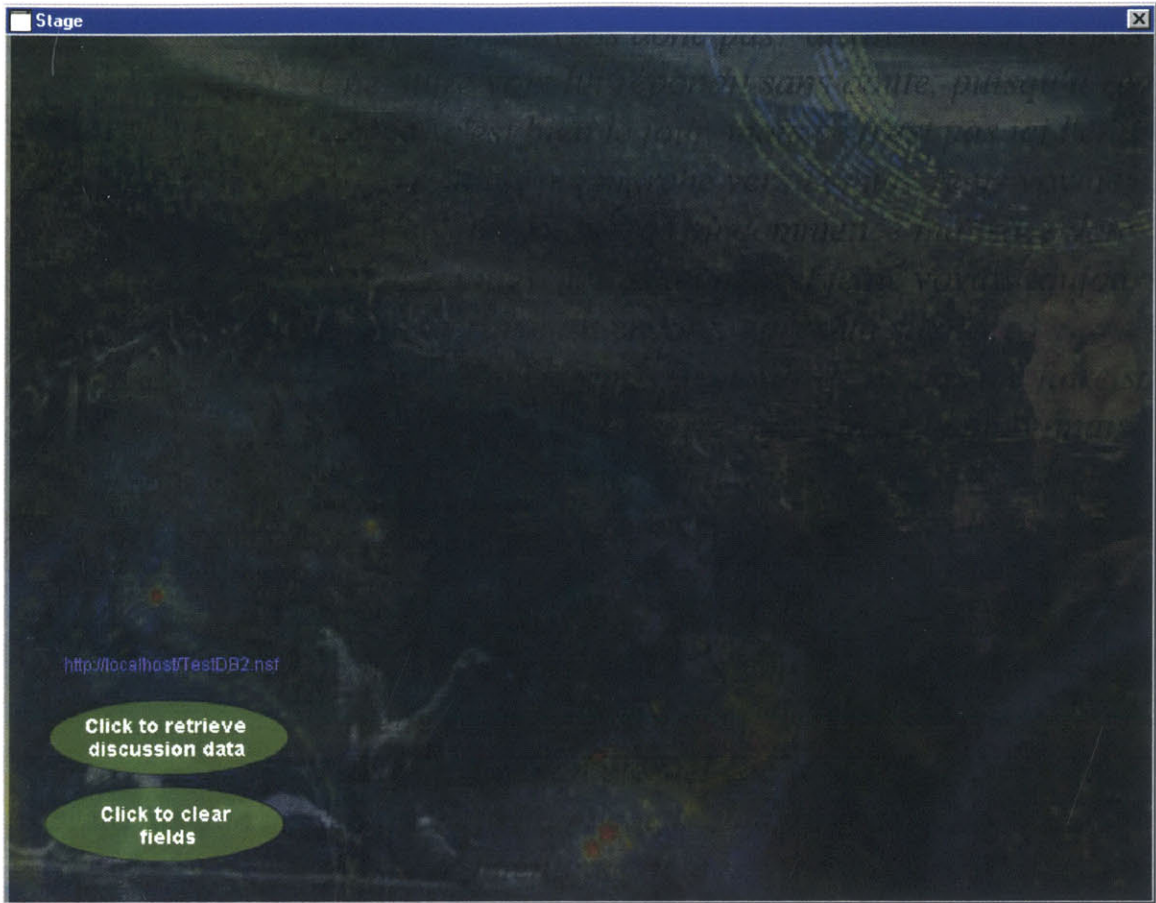


Figure 4.7.2. Prototype in initial state.

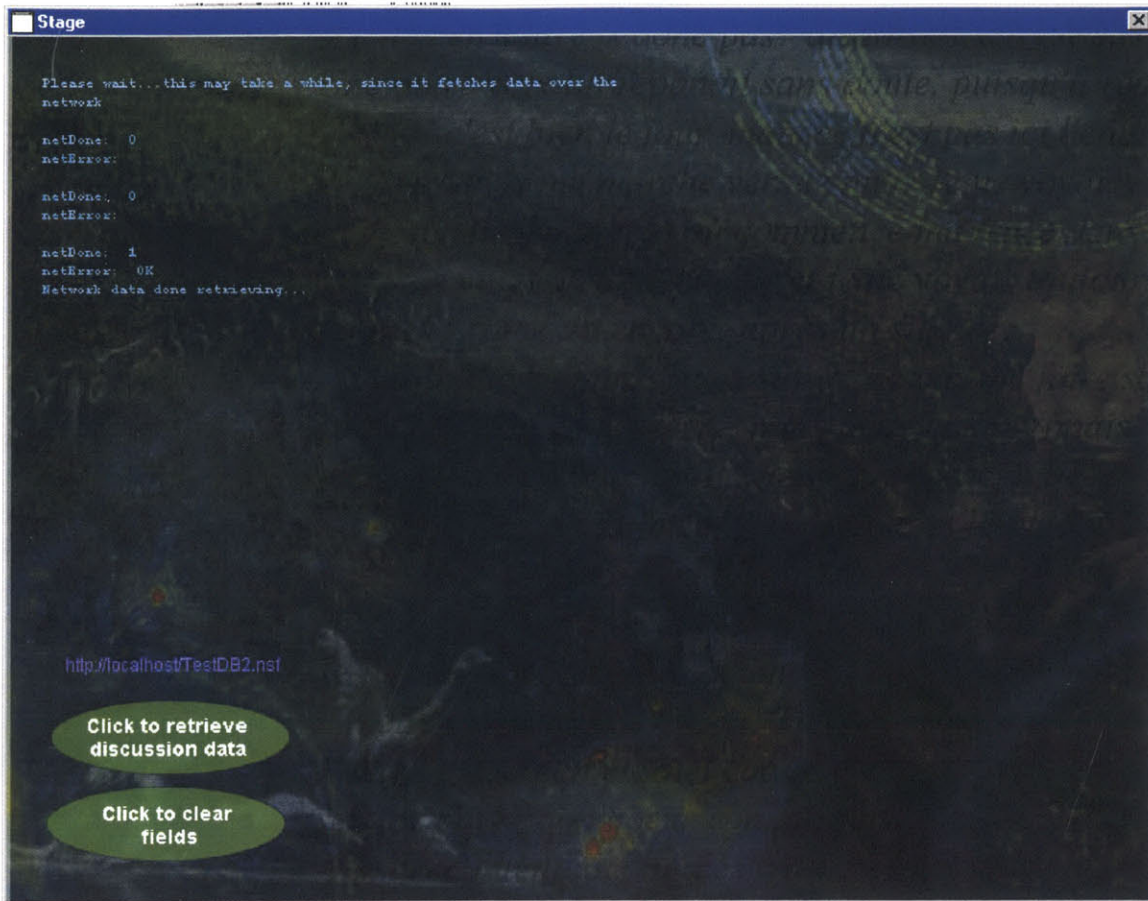


Figure 4.7.3 Prototype upon completing data retrieval over the network.

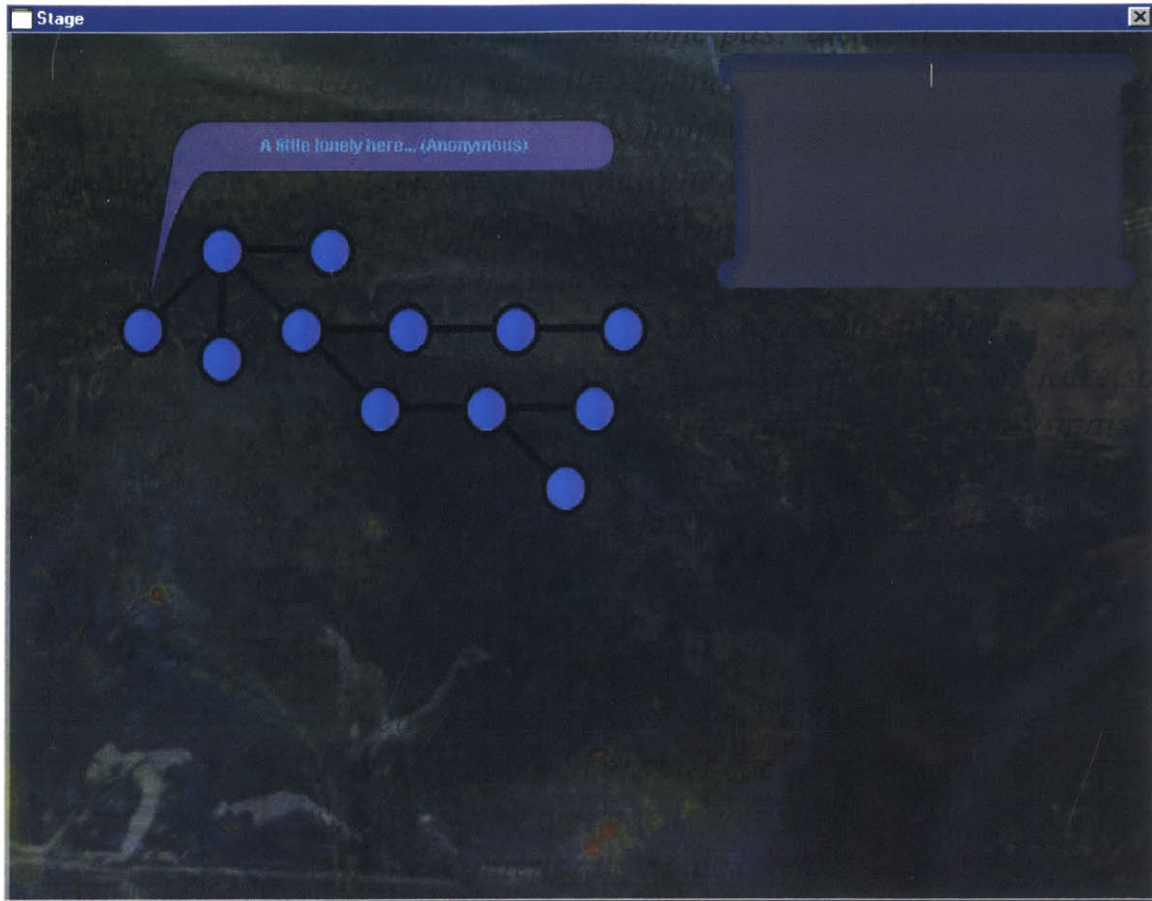


Figure 4.7.4. Prototype with Level Three visualization and a hover-over to show entry title.

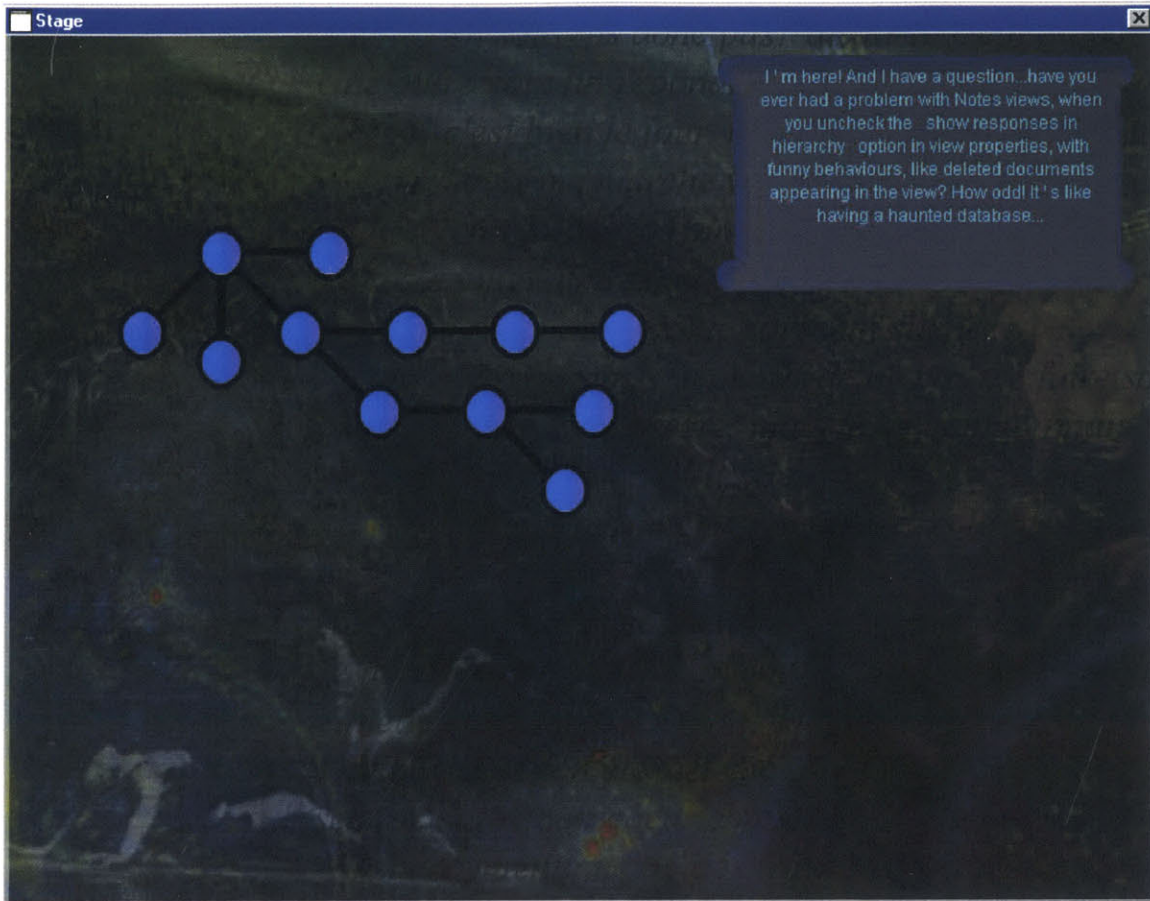


Figure 4.7.5. Prototype displaying entry data after a node has been clicked.

Chapter 5: Discussion

5.1 Results and Issues

We have been able to come up with visualizations which successfully represent various abstract concepts related to online discussion databases. Our representation partitions the data into three levels of detail, and provides an interactive visualization for each level. Although we have not done any formal usability or user testing, our demos have had some exposure to various researchers and product teams within both the MIT and Lotus/IBM communities, and has been extremely well received by those who have viewed it. The visualizations and interface are seen as a step in the right direction in improvement of discussion databases by everyone, and the representations which we designed are comprehensible by most. Most viewers felt that our visualization and interface would be a significant improvement and work well to help browse the kinds of discussion forums they had personally interacted with in the past. There was universal acceptance to the idea of indicating "problem" threads apart from "opinion" threads. It was evident that people unanimously wanted a working version of our interface to use as soon as possible, with those involved in regular use of threaded discussion expressing extreme interest. We feel we have been successful in our primary goal, demonstrating that further research and development in this area is worth doing.

We have also received some feedback as to what could be done differently, both in the data we chose to represent, and the manner in which we chose to represent it. In Levels One and Two, some people felt that the attributes we chose to visualize, while

interesting and valuable to them, were limited in scope, and perhaps should allow the users flexibility in choosing attributes visualized. This was not a problem in level three, however, since the design already allowed for viewing of alternate data. Also, some people felt as if Level One should have had a greater number of possible symbols to represent a more variety of discussion thread states. The choice of symbols in Level Two was also occasionally questioned. This was partially expected, because we had decided that the importance of ease of use should surpass ease of learning at this level. We knew the design would be slightly less intuitive for this reason, and also because it presented a large amount of unfamiliar meta-data. However, even viewers who experienced the Level Two interface repeated times felt that it was difficult to determine which of the attributes represented were more important data than the rest - it was unclear whether the outermost or innermost elements were more essential. The choice of representation in Level Three was the most intuitive to everyone, perhaps because it uses the familiar link-node map structure which most people have been exposed to before, in other contexts.

In our implementation we demonstrated that Director has great potential as a multimedia browser tool for dynamic data - Director is often used for interacting with static data, and its capability of handling dynamic data was unknown. It not only provided popular visual effects that would be difficult to code from scratch, such as background transparency and fades, but it also provided a browser-independent, relatively compact, multimedia format, dynamically downloadable from the web. Its programmability effectively hooked into the network and retrieved data from an outside source of discussion data. Thus it showed that further exploration of Director similar multimedia software, such as Flash, would be worthwhile.

However, Director also had some problems which needed further investigation, or perhaps some changes to its current capabilities. Among these were performance, which was rather slow for any substantial amount of data. This was due not only to slow network access, but also to the amount of time required to parse and display the data. Some of the functionality listed in the help documents for the XML Parser Xtra did not work, and made XML parsing slower and more cumbersome than necessary. There was also a problem with the creation of new sprites during run-time, which caused the screen to flash. Hopefully, as the functionality of Director continues to expand in upcoming versions, some of these side effects will be eliminated, or easier to bypass.

5.2 Related Work

There has been other work which visualizes social interactions on the web, including online discussion forums. This includes the work done by Danyel Fisher at the Department of Computer Science in Berkeley, California [6]. Fisher's work focuses on discussion databases as a medium for creation of interesting social networks, and his work aims to visualize these networks. Warren Sack of the MIT Media Laboratory also visualizes discussion threads; his work focuses on representations of discussion database social networks and semantic processing of natural language[7]. The existing work which is most closely related to ours is the project WebFan, created by Rebecca Xiong, also at MIT [8]. Her work visualizes threaded discussions, but rather than focusing on efficiency of use, her work allows investigation of social interactions, such as how one determines who is an expert in a on-line community. Other similar work has also been done in the

Sociable Media Group at the MIT Media Laboratory, such as Loom, under Judith Donath [9]. The loom project visualized the tone of conversation as a thread progressed, indicating where comments were supportive or contradictory to the thread topic. Some other projects of the Social Media Group attempt to visualize synchronous chat interactions, rather than the asynchronous discussion forums. This body of existing work contains interesting visualizations of social interactions, as well as some retrieval of additional data, such as keyword trends. However, we have uniquely chosen to focus on helping users handle information overload, and reach the information they are looking for more effectively.

5.3 Future Directions

As stated earlier, the scope of our project was potentially overwhelming. As a result, there is much work to be continued in this area.

The idioms of expression and interactions which we have created can be fine-tuned and redesigned further, an exploration which could go on indefinitely. It would also be possible to experiment with other interaction techniques, such as true zooming, and expand our designs to include tools for posting as well as the ones we have proposed for viewing. Since our implementation focused on only part of our design, we did not take advantage of some additional features of our tools, which could be utilized to provide additional features to our prototype. This includes functions such as the `postNetText` command in Director, which allows data to be sent back to the database. There are also different implementation tools, such as Flash, which could be explored as alternatives to

our current choices. In addition, we have not experimented with different or new types of meta-data in the back-end, and entire projects could be based on the work needed to create discussion databases which collect and store all of the meta-data we have suggested.

A goal of our project was to create representations which would effectively represent the average discussion data which exists on the web, but visualizations that could also be tested against greater amounts of real data. Using the results of this testing, both the representations and the interface could be further enhanced or redesigned to withstand data extremes.

Finally, as with any visualization and interface, our proposed designs should eventually undergo usability and user testing. Since we aimed help the users handle the information overload available in discussion databases, user feedback is invaluable for fine-tuning and optimizing our designs

5.4 Acknowledgements

This project would not have happened without my thesis advisor, Julie Dorsey, who instructs the Visualization class which made me realize it would be possible to make this work into a thesis. I would also like to send my sincere thanks to all of the staff at Lotus Research for their support and contributions to this work, particularly Li Cabrera, who went far beyond ensuring I had the computing means to do this work, and Steve Rohall, for his technical guidance. I must also acknowledge my fellow intern, Hyun-Yeul Lee of Carnegie Mellon University, for her enthusiasm, support, and input throughout the

design section of the project. Most of all, I cannot express nearly enough gratitude to Paul Moody, my supervisor for this project, who has given me such incredible inspiration, input, and guidance throughout this work.

References

1. Howe, Denis. The Free Online Dictionary of Computing, 1993.
<http://wombat.doc.ic.ac.uk/foldoc/index.html>
2. Bederson, B. B. Stead, L., and Hollan, J. D. Pad++: Advances in Multiscale Interfaces, *Proceedings of ACM Human Factors in Computing Systems Conference Companion (CHI'94)*, 1994, pp. 315-316
3. Furnas, G.W. Generalized Fisheye views. *In Human Factors in Computing Systems CHI '86 Conference Proceedings*, 1986, 16-23
4. Shneiderman, B. Dynamic Queries for Visual Information Seeking, *IEEE Software*, Vol. 11, No. 6, 1994, pp. 70—77
5. Christopher Ahlberg and Ben Shneiderman. "Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays." *In Human Factors in Computing Systems: Proceedings of the CHI '94 Conference*. New York: ACM, 1994.
6. Fisher, Danyel. Visualizing Social Newsgroup Interaction. 2000.
7. Sack, Warren. "Discourse Diagrams: Interface Design for Very Large-Scale Conversations" to appear in the *Proceedings of the Hawaii International Conference on System Sciences, Persistent Conversations Track*, Maui, HI, January 2000.
8. Xiong, Rebecca. WebFan. <http://graphics.lcs.mit.edu/~becca/webfan/>
9. Karahalios, Karrie. Loom. <http://www.media.mit.edu/~kkaarahal/loom/>