Tangential Browsing

Peter Ree
Advanced Undergraduate Project 6.199
Advisor Glorianna Davenport
February 1999

Abstract

By themselves, current web browsers do not present users with a simple and cohesive means to browse the World Wide Web. Popular search engines such as *Yahoo!* and *hotbot.com* aim to solve this problem by allowing users to search the Web by matching simple queries with related web pages. However, this mechanism adds complexity to the actual act of *browsing* and thus detracts from the intended leisurely nature of the act. Tangent solves this problem by doing much of the work done by users using traditional search engines to browse the Web. It simplifies the browsing procedure by eliminating the formal search engine query while suggesting relevant URLs to the user on its own. However, Tangent is not without problems of its own. Specifically, the immense size and growth of the Internet presents Tangent with difficulties searching for material, as well as maintaining it. However, this paper will show that Tangent succeeds in offering an alternative to using search engines for users that wish to browse the Web in a leisurely fashion.

Introduction

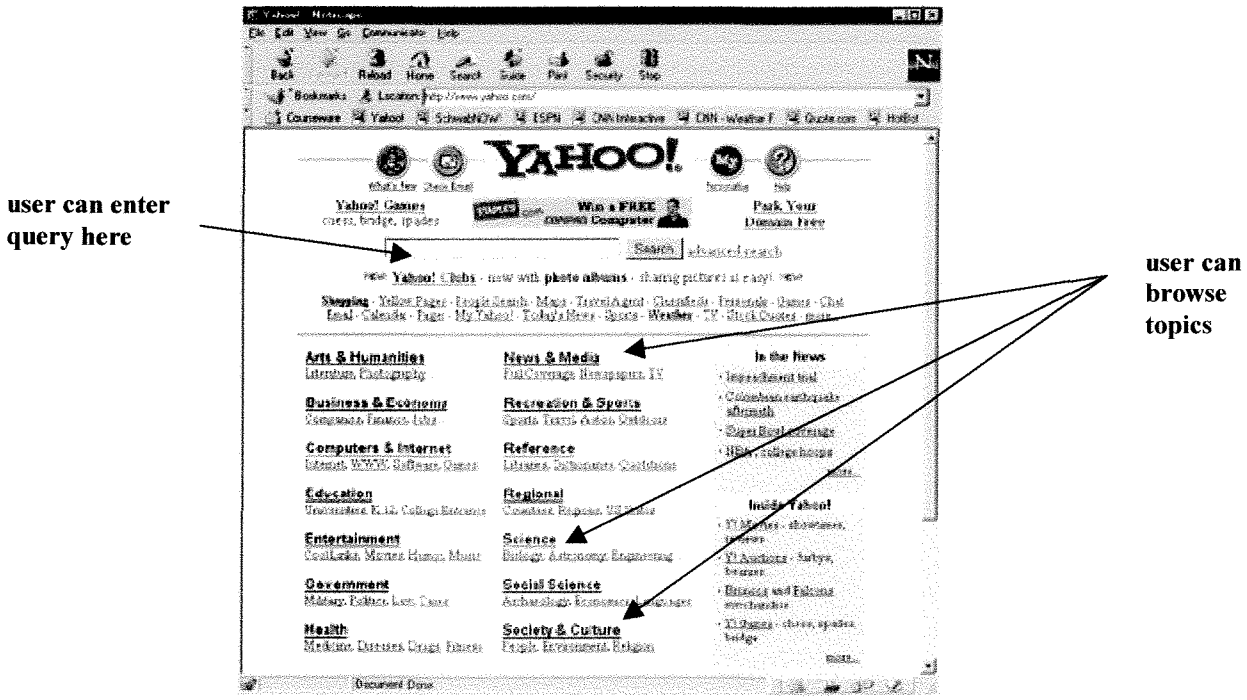
The World Wide Web contains a vast amount of information that spans almost every topic known to mankind. Originally, the Internet was built as an information network for the United States government and was intended to allow key organizations to quickly exchange ideas over long distances. However, with time, we have seen that the increasing number of households, businesses, and other non-government based entities connected to the Internet has led to a more diverse group of users. Consequently the needs of this new diverse body of users has lead to a change in the Internet. Both the actual amount of information available on the Internet as well as its diversity has grown incredibly. No longer is the Internet primarily a data bank of "classical" research. Instead, we see that its diverse content reflects the many different interests and lifestyles of its users. These users are no longer professors or government scientists searching for a specific technical paper. Rather, they are students, children, and parents, of all ages and nationalities. Similarly, we have also seen a change in the behavior of these new Internet users. They often do not use the Internet to search for specific items. Rather, they use it as they would the television. Viewing web pages on the World Wide Web has become for some as leisurely as "channel surfing" with a television. However, I believe that the number of these *browsing* users is only a small fraction of those who would do it if it was in fact as simple "channel surfing".

It is important to note the distinction between leisurely navigating (browsing) the Web and searching the Web. A user *browsing* the Web aims to casually visit various web pages while taking pleasure (or displeasure) in their content. In addition, the user wishes to move on to a new page when desired. Netscape and Internet Explorer have been labeled as "web browsers" by the software industry when in fact they do nothing to help a user *browse* the Web. Aside from the pre-installed bookmarks, both products only offer users a URL address bar to command the "browser" where to go. To those users who wish to actually browse the Web, this is analogous to finding a needle in a haystack without knowing what the needle looks like! Thus, various solutions have been devised. Portal web pages, such as *www.netscape.com* and *www.pathfinder.com*, contain many links to web pages and present them in a organized manner. However, these types of portal pages only encompass a minute fraction of the information available on the Internet. The most common solution and tool that users use to browse the Web is the search engine. However, these Internet mechanisms remain from the time of homogenous users who knew what they were looking for. While we have seen that these engines are an Internet necessity, they have been slow to adapt to changing user preferences. Examining their operation quickly reveals the need for new features or perhaps an entirely new system.

If we begin by studying how a typical user would leisurely browse the Web, we immediately see how the search engine fails. This problem arises from the fact that the user must browse the Web in the same manner as if they were searching for a specific topic. He/she must go to a search engine such as *Yahoo!* or *hotbot.com* and enter a topic to be searched for. The search engine then presents the user with a list of URLs. All engines base their success on the accuracy of matching the user's input with the suggested

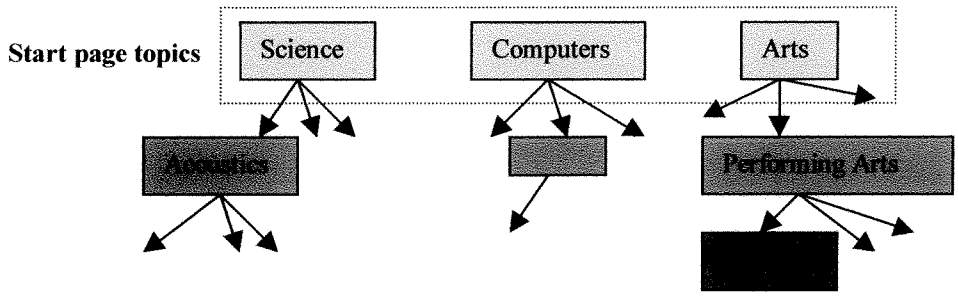
URLs. Should a user wish to browse the Web, instead of searching for something specific, he/she would have to conduct multiple search queries via a search engine. The user would have to think about what he/she is interested in, enter in an appropriate query for the engine, and finally chose one of the suggested links.

Netscape user interface



Not only is this process complex, it is very inefficient. An example would be a Sunday morning Web person who chooses to use the popular *Yahoo!* search engine hoping to browse the Web. This person has two options: entering a query string for *Yahoo!* or clicking on a topic link and continuing to click on sub-topic links until a suitable page has been found. The former method suffers from the fact that a user must continue entering strings once he/she wishes to view a new topic. A weakness in the latter method can also be seen if a user wishes to changes topics. For instance, the user may click these topics in succession: *Arts, Performing Arts,* and finally stops after clicking *Books*. Now, the user in effect has traversed a "topic tree" and reached a node which he / she is interested in. However, if the user now wishes to change topics, say to computers, the process of browsing must start over from the initial *Yahoo!* start page.

A typical search engine "topic tree":



This paper will discuss a design that aims to allow users to browse, not search, the World Wide Web in an efficient and cohesive fashion that is consistent with *browsing*. Key design criteria for a successful system will be outlined and discussed. In addition, a discussion regarding a working implementation of the design criteria (named Tangent) is also included as well as some unexpected findings.

Before continuing on into a discussion of design principles, it is important to note that I have chosen to limit the scope of my research to examining the effectiveness of Tangent in browsing a set amount of URLs. While this does not accurately portray the Web and it's immense number of web pages, it serves as a useful model to help maintain the focus of the research. Instead of also designing a system to connect to URLs via HTTP and load, interpret, and display HTML, existing Web browsing software can be used in cooperation with a browsing component. Thus, this paper will focus on the design of the browsing component and the problem of *browsing*, not retrieving, World Wide Web pages.

Design Criteria

This section will discuss important design criteria in building a system that allows users to browse the World Wide Web. Although somewhat abstract, the design criteria listed below must be implemented in any successful system. Specifically, the criteria are:

The system automatically presents a user with multiple URLs that the user is interested in
The system must present URLs that the user is interested in or is relevant to what the user is currently browsing. Although presenting random material may help discover new interests, the Web's content is too diverse and contains too many pages to simply present random materials. Doing so would decrease the chances of presenting URLs that the user is interested in and would thus make the user worse off compared to using traditional search engines. Consequently, only presenting random material would defeat the purpose of even building the system.

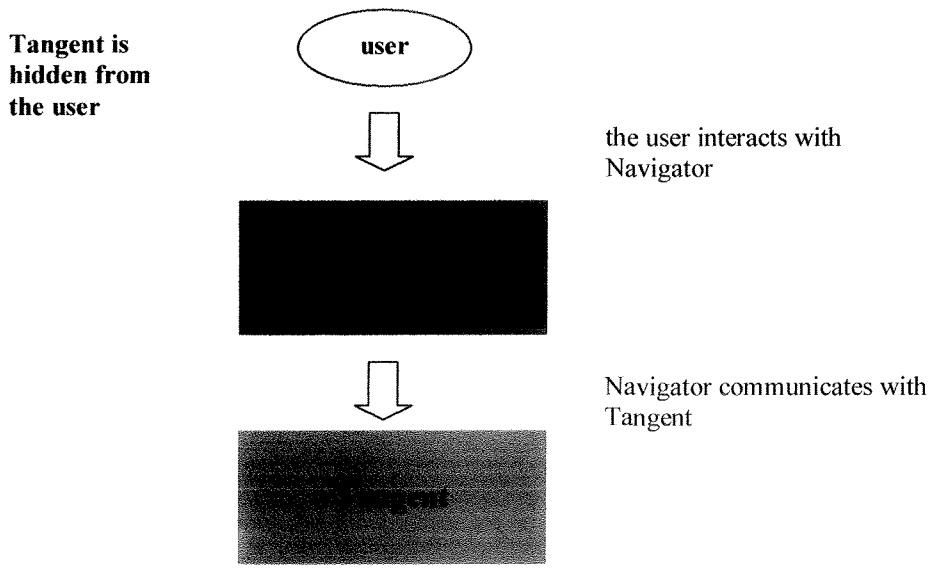
The system is quick to make suggestions
In addition to providing the user with relevant suggestions, the system should be present them to the user quickly, so that the user's focus is on the URL he / she is viewing, not on the system itself. The design should strive to eliminate user interactions with any search engine like interface. That is, use of the system should not require any extensive involvement by the user. For example, using a search engine not only requires the user to pause to think what he or she is interested in, but the user is also forced to try and express these interests in words which must be typed in the search query. Thus

presenting quick suggestions implies transparency of system operations and also minimal user interaction with the system itself.

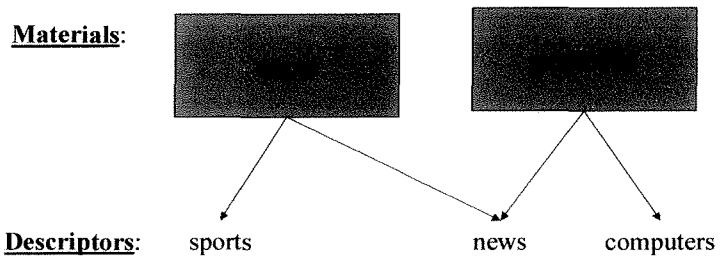
The system is simple to use and comprehend
Similarly, it is also important that the system is simple to use and comprehend so as not detract from the actual viewing of the web page, which is the user's initial goal. A simple to use system implies that the user interface is well constructed, easy to comprehend, and easy to use. This will also help keep the user's focus on experiencing the web page rather than using the system. One would rather focus on the "television show" on the current channel rather than the "remote control" used to get there.

Design Recommendation
This section will outline the general behavior of the Tangent system, a system which implements the 3 key design criteria discussed above. It is important to first note that Tangent is written as a Java applet which allows the system to be simply, and transparently loaded by current commercial web browsers (such as Internet Explorer 4.0) via the Internet. This has allowed the research to focus on browsing, rather than retrieving, web pages as discussed in the introduction. However, using an applet has introduced some restrictions to the design that will be later covered in a more rigorous discussion of Tangent. Once Tangent is loaded, images corresponding to specific URLs are displayed. Moving the mouse pointer over one of these images causes the corresponding URL address to be displayed on the interface. This conveys that the cell represents that web page. Clicking on one of these images will cause the corresponding URL to be displayed in the web browser and will also cause Tangent to suggest new URLs by updating the URL/image choices available on the interface.

Design Overview
The design of Tangent can be viewed as two independent components: the user interface and the selection system. It is actually the selection system that is known as and will from here on be referred to as *Tangent* while the interface will be referred to as *Navigator*. The user has no direct access to Tangent and thus is hidden from its complex features. Instead, the user can use the simpler Navigator interface to indirectly use Tangent.



Tangent
Tangent functions on the notion that a user's decision to view a URL is based on evaluating two key elements. These elements are the URL itself, and the set of attributes that describe it. A person subconsciously evaluates the attributes of a given web page. Specifically, he/she examines the topics a URL contains and decides if it is worth viewing. Tangent uses two important objects to encapsulate the abstract notions of web page content and content topics. *Materials* are used to represent URLs while *descriptors* are used to represent a description of a material's content. Thus each unique material has an associated set of descriptors.



When a user chooses to view a material, Tangent can extract the descriptors associated with this decision. The system records these descriptors as being viewed by the user. As the user continues to use the system to browse material, the system can build a set of descriptors that tend to be associated with materials that the user chooses to view. This set can be viewed as the user's interests. Thus the system is able to make relevant material suggestions by suggesting materials that share descriptors with the user's interests.

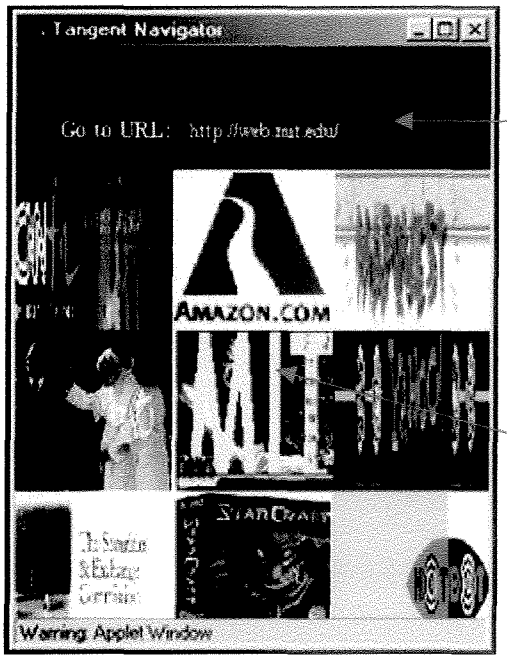
Navigator
The user interface consists of a grid of nine squares where each corresponds to a URL. A given cell in the grid may contain multiple images. The number of images is dependent on the number of descriptors a material contains and likewise each image should reflect one of the descriptors. For example, the

descriptor *basketball* may have an image of Michael Jordan for the URL www.nba.com and an image of Lisa Leslie for www.wnba.com. Each image is displayed in succession. Moving the mouse over a cell causes rate of the images to be displayed faster. In addition, the corresponding URL is displayed at the top of the interface.

It is intended for the user to interpret each image as describing the corresponding URL. Using visual cues instead of text based cues (such as displaying the word "basketball" for the descriptor *basketball*) helps to expedite the user's decision of what URL to chose. In addition to saving time by using images, displaying descriptors as text may be somewhat vague. The WNBA and NBA example shows that a single image can immediately inform the user what league the image corresponds to whereas the word "basketball" is ambiguous.

The speed up effect of moving the mouse over a cell is done for a user who wishes to quickly cycle through the URL's images. In addition, it alerts the user that the mouse pointer has some significance and that he/she will be transported to that cell's URL if the mouse is clicked.

The Navigator user interface



clicking the mouse will take the user here

cell containing a URL

Implementation Details

Tangent

Descriptors
Each descriptor is considered unique and treated as a single object. Thus there is only one *basketball* descriptor and it is used to describe www.wnba.com as well as www.nba.com. Each descriptor object keeps track of how many times it has been activated as a consequence of being associated with a selected material. In addition, each descriptor has a unique identification value which is implemented as a positive integer value that is a power of 2. This allows the ID number to appear as a single 1 in a string of bits (where all other bits are 0s). Descriptors also contain a flag to specify weather or not the user is interested in the descriptor.

Example:
name: *basketball*
ID number: 4 (....00100)
times activated: 2
interest?: no

Materials
A material contains its name, the URL it correspond to, an array of descriptors that describe it, an array of pictures to display, and a descriptor set value.

Example:
name: nba
URL: www.nba.com
times activated: 2
interest?: no

The descriptor set value is derived by summing the ID values of each individual descriptor in the material's descriptor array. Thus we can see why it is important to have the descriptor identification values be powers of 2.

For example, if a material is described by 2 descriptors *sports* and *news*:

sports identification value = ...00001
news identification value = ...00010
(the "...." is to denote an arbitrary number of 0 bits preceding the sequence")
set identification value = ...00011

Notice that the existence of a descriptor is maintained even after summing many identification values. Thus examining this value will reveal precisely all descriptors of a material.

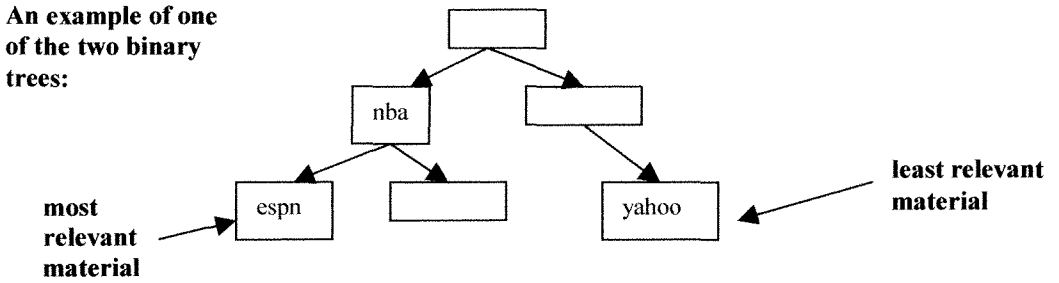
When a user chooses to view a material, Tangent immediately loads the material's profile (its URL, picture array, etc.). Most importantly, Tangent retrieves the material's descriptor set and sets it to be the current "active" set. This descriptor set represents current user interests and allows Tangent to fetch similar materials. In addition, Tangent increments the "viewed" counter in each descriptor in this set to reflect that

the user is viewing a material that the descriptor describes. If the counter in any of these descriptors is above a threshold value, the descriptor is set to being an "interest" of the user. This threshold value is set when the Tangent system is instantiated. Setting a value of 8 will tag all descriptors that have been viewed 8 or more times as a user's interest, and all descriptors viewed less than 8 times as non-interests. All descriptors with the interest flag compose a second descriptor set, the "interest" set.

All materials in the system can be searched based on either the active or interests descriptor sets. From the active set, the system can return materials that are relevant to what the user is currently. Likewise, the interests set allows it to return materials that reflect what the user has tended to be interested in.

Operation

In order to search for relevant materials, the system creates two values corresponding to each the sets (interest and active). These values are derived in the same manner as a material's descriptor set value. All descriptors in a set are summed. Thus one single value can represent current active descriptors while another can reflect all user interests. Two binary trees are then formed, one for each set. Each tree is formed by taking one of the set values and comparing it to each material in the system. Specifically, each material's descriptors value is compared to one of the set values for similarities. From these comparisons, Tangent can create a binary tree based on a material's similarity with one of the two user descriptor sets.



Navigator

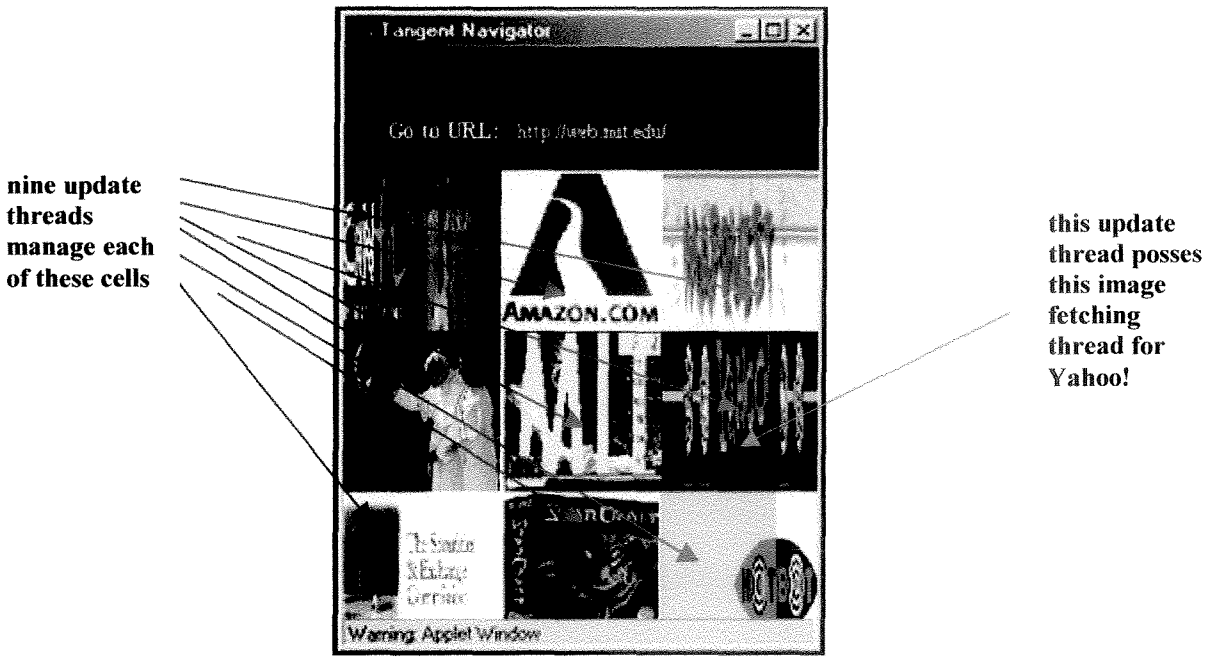
The entire system consists of three main components: the applet, a navigator frame, and a Tangent engine. The applet serves as a starting point for all processes to begin and also serves as the system's link to the browser. The applet is responsible for executing all requests to load images as well as showing URLs in the web browser. The navigator frame, referred to as "Navigator" from here on, is the actual user interface and displays the 9 images. The Tangent engine, whos operation was discussed above, is responsible for making suggestions. Navigator acts as a layer between the user and the actual Tangent system and allows simple user actions such as mouse clicks to trigger complex behaviors.

Component	Responsibilities
Navigator	Interacts with the user, responds to user actions Displays images, changes images appropriately Passes user selections to Tangent Commands the applet to show appropriate URL
Tangent	Maintains and records user's interests. Returns a list of most active/interested materials to Navigator
Applet	Maintains a link to the HTML page that contains the applet

Loads URLs for the user to browse, as well as fetches images in the background

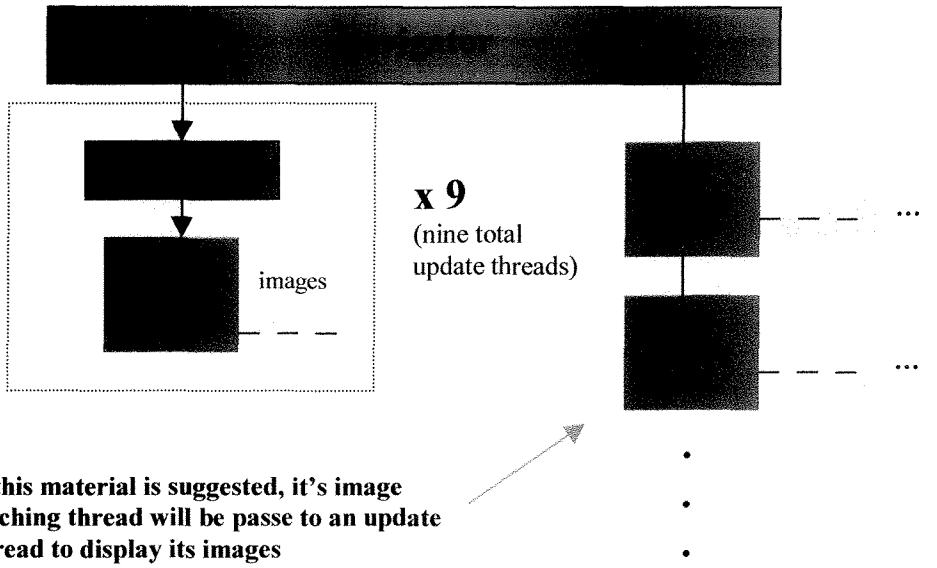
However, in order to fully understand how the user interface operates, it is important to examine how it initializes itself.

A special class (discussed later) is used to load information via the Web to initialize the Tangent engine with the proper set of materials and descriptors. Once this is done, the Tangent engine is ready to be used and the Navigator frame can now be initialized. Before the frame is displayed, image fetching threads are created for each material in the Tangent. These threads are responsible for fetching images over the Internet. Nine update thread are also spawned and have the responsibility of drawing images to the Navigator user interface. Each of the nine threads corresponds to a single cell in the grid. Thus each of the nine will also have an image fetching thread that corresponding to the material represented in a specific cell.



When a user clicks on a material, the Tangent engine is consulted and the most relevant materials are retrieved from it. With this list of materials to display, Navigator (as the Java class is called), retrieves the image fetching threads responsible for each of the materials. These threads are then passed on to the update threads which can now draw these new images to the display.

Navigator and it's threads



Initializing Classes

The special Tangent initializing class mentioned above allows a simple text file to fully define the materials and descriptors of a new Tangent engine. This file can be placed on the Internet as a web page and thus allows the engine to be remotely initialized. This saves space in that the applet class files do not have to fully embody the state of the Tangent. This also allows for many different configurations of Tangent to be readily available.

Implementation Efficiency

It is important to note the reasons for choosing the before mentioned implementations for Tangent and Navigator. Specifically, it is important to examine the efficiency of the algorithm responsible for searching for relevant materials. This process involves searching a binary tree that grows logarithmically with respect to the number of materials in the system. This feature is quite nice given that a poorer algorithm may have involved sequentially searching through all materials and would thus suffer from orders of growth greater than logarithmic. The choice of implementing descriptor identification values as single bits further optimizes the algorithm. A material's descriptor set can be expressed a single value that can be compared to another single value (the interest or active descriptor set/value). This comparison only involves right bit shifts and bitwise AND operations, both are *very* computationally cheap. This aids in the speed and efficiency of constructing the two binary trees.

The interface also benefits from a good internal design in its efficient use of time sharing operation. That is the multithreaded nature of the image fetching allows multiple images to be loaded at the same time. If one image is taking long to load, other images can still be loaded. It is also possible for the Navigator to be displayed without completely loading all images. When a user is viewing a URL, in other words not interacting with Navigator, the image fetching threads can still be retrieving images that have not finished

loading. Navigator gracefully handles the problem of a missing, or unloaded image by simply not drawing the image to the screen. This is rather different than the approach of showing a broken image or an image with a question mark that most web browsers chose to do in such a situation

However, it is also important to note some implementation shortcomings that lead to inefficient behavior. The most outstanding weakness in the system is its implementation as an applet. The current restrictions of an applet do not allow it to connect to any other computer than the web server it came from. Thus, Navigator's image fetching threads can only fetch images from the web server hosting Tangent and thus all images must be stored on the web server. This includes visiting the web page and copying the image to the web server. It would be nice to simply specify the URL for the appropriate image, this cannot be done now. However, in the future, with the prospect of signed applets and weaker applet restrictions, this is a definite possibility. The system could have been implemented as a unique and proprietary client and would thus have very little restrictions. However, this would have greatly increased the development overhead and would most likely deteriorate the spirit of the research.

Criteria Satisfaction

The choice of implementation of a browsing system satisfies the three criteria specified earlier.

The system automatically presents a user with multiple URLs that the user is interested in
All user selections of URLs are recorded in Tangent and thus allow it to suggest URLs that have descriptions that the user has tended to be interested in (from past history) or have similar descriptions with the user's current selection.

The system is quick to make suggestions
The efficient creation of the binary trees, the searching of the trees, as well as the image fetching threads all allow the system to quickly make suggestions to the user.

The system is simple to use and comprehend
The use of images to convey description requires little explanation to the user. In addition, the "flicker" of the images when the mouse pointer is over a cell immediately informs the user that the mouse pointer has significance when using Navigator. The only interaction a user must do in order to use the system is to click on one of the 9 cells. The text "Go to URL: " is displayed as well as the URL corresponding to the cell the mouse is currently over. Suggestions as well as updating the graphical interface is all handled by the underlying system and thus appears transparent to the user. In addition, the interface does not clutter the display by showing broken or question mark images when an image is not loaded or is missing. This helps to preserve the aesthetic "tranquility" of the interface.

Design Weaknesses
The chosen design exhibits some inherent weaknesses that will be discussed in this section. Specifically, Tangent has a very weak mechanism for filtering out descriptors that the user is not interested in, but selects to view. This situation can come about if a user wishes to view a URL, but then decides he/she is not interested in its content or perhaps he/she accidentally selected the URL. The only means Tangent has for guarding against marking these viewed descriptors as the user's interests is by use of the threshold value. Thus Tangent relies on a user choosing to view materials with descriptors he / she is interested frequently while viewing uninteresting materials infrequently. This presents another problem in that a suitable threshold value must be chosen. This is difficult for a variety of reasons. One significant problem is that users have different behaviors and will thus tend to browse differently. For instance, one user may behave "nicely" and select materials that he/she is interested in. However, a different user may chose a random material from the 9 cells to view. It is difficult to determine what the threshold value should be for the random user or any other user close to this type. The best solution would be to assign threshold values specific to each user. However, there is no simple means to do this. One possible solution that I

considered was allowing the user to inform Tangent if they were pleased with the URL after viewing it. However, I believed that this added an unnecessary and somewhat cumbersome user-interface interaction.

Another weakness in the design is the fixed amount of material a user can see at any given time. Currently, Navigator shows 9 materials, but what if a user would like to see 16 or 4? This again stems from the problem of differentiating user behaviors. One solution would be to allow the user to specify an image "bandwidth" value and would cause Navigator to have more or less image update threads accordingly. However, I felt that research into this matter and its usefulness should be done at a later time. For the purposes of this project, I found that the 3x3 matrix worked very well.

An interesting weakness that I discovered actually helped to improve my system. This particular design flaw occurred in cases where many materials were closely related, that is, choosing to view one of these materials would trigger Tangent into suggesting the others. Thus it is possible for Tangent to only suggest these materials because choosing one of them does not lead to any materials outside of this group. I called this a *state trap* during my research and found that the best way to avoid this was to insure that random materials were always part of Tangent's suggestions. This in turn, helps the user explore new topics and thus makes Tangent a more versatile system.

Conclusion

If time permitted, I would have liked to shown my system to more people and added more URLs to the Tangent initialization file. However, my findings were very positive. Although the design suffers from the before mentioned weaknesses, I found that it succeeded in improving the leisure nature of browsing the Web. I found that it replaced the complicated and time consuming process of using *Yahoo!* with a simple and somewhat entertaining mechanism. This novel value of actually using Navigator, however, strayed from the original goal of maintaining the user's focus on the URL, not the browsing mechanism. I found that I enjoyed interacting with Navigator more than I did viewing the URLs. This however, is probably due to my heavy involvement with building Navigator and familiarity with the links. I am confident that this appeal would most likely decline with further use of the system.

It is important to also note the feasibility of actually implementing a real world Web browsing Tangent powered system as outlined in this paper. This would require many new additions to the current design as well as changes to current and popular technology. One problem I discovered was that, surprisingly, most web pages do not have many relevant images. That is, the images they use often do not reflect the descriptors that were used to describe them. Thus, I resorted to using images that did not reflect a descriptor. Often times, I found that I had to use business logo images when a suitable image could not be found. Another problem in attempting to make a real browser capable of browsing all Web links is that Tangent was designed to handle a fixed amount of content. The fixed amount of content allows us to define each material and each descriptor. The real Web represents a much more complex and difficult problem. There is no simple way to correlate each URL on the Web with a set of descriptors and it is even more difficult to compress all of this information into a single applet. However, I believe that this will be soon possible with new, soon to be released technologies. One could imagine building a server back end to "crawl" the web as traditional search engines do. During it's traversal of the Web, the server could examine XML pages and extract relevant descriptors and images. This information could then be passed to Tangent as needed. While this quick sketch of a design leaves out too many details to actually be implemented, it does show that we are not too far away from making a Tangent browser a reality. In fact, the latest version of Netscape already includes a simple mechanism to help users browse. The Netscape menu bar now contains a "What's related" button that will retrieve links related to the URL the user is currently browsing.

Although the Netscape mechanism is somewhat limited, it reveals how users should expect to browse the Web in the future. Soon, Web users will have a simple mechanism that will facilitate leisurely and simple browsing of the World Wide Web. I hope that the preliminary research I have done on this subject will

give some insight to what browsing will become. Should it not, however, there is no doubt in my mind the project was a success. I have learned much more than I originally intended. Specifically, I learned many lessons in software development and project management that I did not expect to learn. In addition, the very thought provoking nature of the project made the entire project worth working on. In the future, I know I will be able to apply the lessons I have learned while working on this project, I only hope that I will have a chance to continue my research. Until then, I hope that I have revealed the interesting aspects about Tangent and *browsing* as it was meant to be.

```
public class BTree {

 /* Overview: Class BTree is a data structure used that represents a binary tree.
        It contains methods for adding nodes to the tree as well as methods
        to manipulate a given node's attributes.
 */

 private BTree left;
 private BTree right;
 private BTree parent;
 private Material head;
 private int val;
 private boolean traversed = false;

 public BTree(Material head, int intersection, BTree parent){
  /* requires: head is not null
     modifies: this
     effects: sets this.head = head, val = intersection, this.parent = parent
     */
  left = null;
  right = null;
  this.parent = parent;
  val = intersection;
  this.head = head;
 }

 public void addNode(Material newNode, int newVal){
  /* requires: newNode is not null
     modifies: left or right
     effects: adds a new node to the left branch of this if newVal is ≥ to this.val
              else adds it to the right branch of this
     */
  if(newVal ≥ val){
   if(left == null)
    setLeft(newNode, newVal);
   else
    left.addNode(newNode, newVal);
  }
  else {
   if(right == null)
    setRight(newNode, newVal);
   else
    right.addNode(newNode, newVal);
  }
 }

 public void setLeft(Material newLeft, int newVal){
  /* requires: newLeft is not null
     modifies: this
     effects: sets this.left to point to a new node containing newLeft and newVal
     */
  left = new BTree(newLeft, newVal, this);
 }

 public void setRight(Material newRight, int newVal){
  /* requires: newRight is not null
     modifies: this
     effects: sets this.right to point to a new node containing newRight and newVal
     */
  right = new BTree(newRight, newVal, this);
 }

 public BTree getLeft(){
  /* requires:
     modifies:
     effects: returns the pointer to this' left branch
     */
  return left;
 }

 public BTree getRight(){
  /* requires:
     modifies:
     effects: returns the pointer to this' right branch
     */
  return right;
 }

 public Material getMaterial(){
  /* requires:
     modifies:
     effects: returns a pointer to head
     */
  return head;
 }

 public int getVal(){
```

```
  /* requires:
     modifies:
     effects: returns val
     */
  return val;
 }

 public BTree getParent(){
  /* requires:
     modifies:
     effects: returns a pointer to this' parent
     */
  return parent;
 }

 public boolean wasTraversed(){
  /* requires:
     modifies:
     effects: returns traversed
     */
  return traversed;
 }
 public void makeTraversed(){
  /* requires:
     modifies: traversed
     effects: sets traversed to true
     */
  traversed = true;
 }

}
```

```
public class Descriptor {
 /* Overview: this class is used to contain information about a descriptor that
          describes a material. descriptors are unique.

   */


 private String name;
 private int hits;
 private boolean interested;
 private long ID;

 public Descriptor(String name, long id){
  this.name = new String(name);
  hits = 0;
  interested = false;
  ID = id;
 }

 // selectors
 public String getName(){
  /* requires:
     modifies:
     effects: returns a pointer to this name
     */
  return name;
 }

 public int numHits(){
  /* requires:
     modifies:
     effects: returns hits
     */
  return hits;
 }


 public boolean isInterested(){
  /* requires:
     modifies:
     effects: returns intersted
     */
  return interested;
 }

 public long getID(){
  /* requires:
     modifies:
     effects: returns ID
     */
  return ID;
 }

 // modifiers
 public void hit(){
  /* requires:
     modifies: hits
     effects: increments hits by one
     */
  hits++;
 }

 public void fade(){
  /* requires:
     modifies: hits
     effects:  decrements hits by one
     */
  if(hits > 0)
   hits--;
 }

 public void makeInterested(){
  /* requires:
     modifies: interested
     effects:  sets interested to true
     */
  interested = true;
 }


 public String toString(){
  /* requires:
     modifies:
     effects: returns a string representation of this
     */
  String str = new String("Descriptor:\t"+name+"\n");
  str+="hits:\t\t"+hits+"\n";
```

```
    str+="interested:\t"+interested+"\n";
    return str;
  }

}
```

```
import java.net.*;
import java.io.*;
import java.applet.*;
import java.awt.Color;

public class FinalApplet extends Applet {
 /* Overview: The starting point for everything. Initialized the GUI and the Tangent
          engine
 */
 private Tangent engine;
 private Navigator navigator;

 public void init(){
  /* requires:
     modifies: engine, navigator
     effects:  creates a new Tangent and a new Navigator
     */

  // intialize tangent
  setBackground(Color.black);
  engine = new Tangent(1);
  try{
   URL url = new URL("http://deepc.mit.edu/Peter/AUP/initFiles/init2");
   //URL url = new URL(http://deepc.mit.edu/Peter/AUP/initFiles/tangent.init);
   InputStream  is = url.openStream();
   System.out.println("stream opened");
   InitTangent creator = new InitTangent(is, engine);
   creator.init();
   System.out.println("tangent intialized");

  }
  catch(MalformedURLException e){
   System.out.println("malformed url: "+e.getMessage());
  }
  catch(IOException e){
   System.out.println("IO: "+e.getMessage());
  }

  // create navigator
  navigator = new Navigator(engine, this);

 }

 public void start(){
   /* requires: navigator is not null
     modifies:
     effects:  displays the navigator's GUI to the user
     */
   // start up Navigator GUI
   navigator.show();
 }

}
```

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.net.*;

public class ImageFetcher extends Thread {
 /* Overview: A class that used to fetch images over the web. Contains methods
          for an image's loaded status


 */

 //private Applet dummyApplet;
 private Material m;
 private MediaTracker mt;
 private Image[] images;
 private boolean started = false;
 private int init;
 private Toolkit tk;

 public ImageFetcher(Material m, Frame parent){
  tk = parent.getToolkit();
  this.m = m;
  Vector imageNames = m.getSurfPicts();
  init = imageNames.size();
  images = new Image[init];
  mt = new MediaTracker(parent);

  for(int i=0; i<init; i++){
   try{
    // start fetching images
    images[i] = tk.getImage(new URL((String) imageNames.elementAt(i)));
   }
   catch(MalformedURLException e){
    System.out.println("malformed URL: "+e.getMessage());
   }
   mt.addImage(images[i], i);
  }
 }


 public void run(){
  /* requires:
     modifies: mt
     effects: loops until all images are loaded or until one image fails to
              be loaded
   */
  try{
   for(int i=0; i<init; i++)
    // wait for the images to be completely loaded
    mt.waitForID(i);
  }
  catch(InterruptedException e){
   System.out.println("image fetch interrupted");
  }
 }

 public synchronized boolean isLoaded(int id){
  /* requires:
     modifies:
     effects: returns true if image located at id in mt is completely loaded
     */
  if(id<init)
   return mt.checkID(id);
  else
   return false;
 }

 public synchronized Image getImage(int id){
  /* requires: 0≤ id < images.size
     modifies:
     effects: returns a pointer to the image images[id]
     */
  return images[id];
 }

 public Material getMaterial(){
  /* requires:
     modifies:
     effects: returns m
     */
  return m;
 }

 public int numImgs(){
  /* requires:
     modifies:
     effects: the number of images this is responsible for fetching
     */
```

```
    return init;
  }
}
```

```
import java.net.*;
import java.io.*;
import java.util.*;

public class InitTangent {
 /* Overview:  this class can instantiate a new Tangent object from a given
               input stream that adheres to a specific syntax


          <material name> <material url> {
          <descriptor1> <descriptor1 pict url>
          <descriptor2> <descriptor2 pict url>
          .
          .
          .
          .
          }
 */


 private InputStream initStream;
 private Tangent initTangent;


 public InitTangent(InputStream initStream, Tangent blankTangent){
  this.initStream = initStream;
  initTangent = blankTangent;
 }

 public void init(){
  /* requires: initStream and initTangent are not null
     modifies:
     effects:
     */
  Hashtable dTable = new Hashtable();
  StreamTokenizer st = new StreamTokenizer(initStream);
  st.wordChars(33, 126);
  String name, url, curD, curDP;
  Material m;
  Descriptor d;
  Vector descriptors;
  Vector descriptorPicts;
  long dCounter = 1;

  String term = new String("}");

  try{
   while(st.nextToken() != st.TT_EOF){

    // first token is a word - name of material
    name = st.sval;
    //System.out.println(name: +name);

    st.nextToken();
    //System.out.println(url: +st.sval);
    // next token is the material's url
    url = st.sval;

    // skip '{'
    st.nextToken();

    descriptors = new Vector();
    descriptorPicts = new Vector();

    st.nextToken();

    while(!term.equals(st.sval)){
     // read in all the material's descriptors and picts
     curD = st.sval;
     st.nextToken();
     curDP = st.sval;
     //System.out.println(d name: +curD+ id: +dCounter+ url: +curDP);
     // need to lookup for existance first
     if(dTable.containsKey(curD))
      d = (Descriptor) dTable.get(curD);
     else{
      d = new Descriptor(curD, dCounter);
      dTable.put(curD, d);
     }
     descriptors.addElement(d);
     descriptorPicts.addElement(curDP);
     dCounter<<=1;
     st.nextToken();
    }

    try{
```

```
        m = new Material(name, url, descriptors, descriptorPicts);
        initTangent.addMaterial(m);
       }
       catch(TangentException e){
        System.out.println("Tangent: "+e.getMessage());
       }
       catch(MalformedURLException e){
        System.out.println("malformed url: "+e.getMessage());
       }

     }
    }
    catch(IOException e){
     System.out.println("IO: "+e.getMessage());
    }
  }

}
```

```
import java.net.*;
import java.util.*;

public class Material {
 /* Overview: This class is used to represent a material in the Tangetn system
              a material reflects a URL on the web an has an associated set of
              descriptors that describe it. In addition, it has pictures used
              to represent it
  */


 private String name;
 private URL url;
 private Vector descriptors;
 private Vector surfPicts;
 private long descriptorIDs;


 public Material(String name, String URL, Vector descriptors, Vector pictNames) throws MalformedURLException{
  this.name = new String(name);
  this.url = new URL(URL);
  this.descriptors = (Vector) descriptors.clone();
  surfPicts = (Vector) pictNames.clone();
  Enumeration e = this.descriptors.elements();
  descriptorIDs = 0;
  while(e.hasMoreElements()){
   Descriptor d = (Descriptor) e.nextElement();
   descriptorIDs+=d.getID();
  }
 }

 // selectors
 public String getName(){
  /* requires:
     modifies:
     effects: returns a pointer to name
     */
  return name;
 }

 public URL getURL(){
  /* requires:
     modifies:
     effects: returns a copy of url
     */
  try{
   return new URL(url.toString());
  }
  catch(MalformedURLException e){
   // should never happen
   System.out.println("Material.getURL caught MalformedURLException: "+e.getM
  }
  // to get around compiler complaints
  return null;
 }

 public Vector getDescriptors(){
  /* requires:
     modifies:
     effects: returns descriptors
     */
  return descriptors;
 }

 public Vector getSurfPicts(){
  /* requires:
     modifies:
     effects: returns surfPicts
     */
  return surfPicts;
 }

 public long getDescriptorIDs(){
  /* requires:
     modifies:
     effects: returns descriptorIDs
     */
  return descriptorIDs;
 }


 // modifiers
 public void associateDescriptor(Descriptor d){
  /* requires: d is not null
     modifies: descriptors
     effects: adds d to the set of descriptors that describe this
     */
  descriptors.addElement(d);
```

```
 }


 public String toString(){
  /* requires:
     modifies:
     effects: returns a string representation of this
     */
  String str = new String("Material: "+name+"\n");
  str+="URL: "+url+"\n";
  str+="Descriptors: ";

  Descriptor d;
  Enumeration e = descriptors.elements();
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   str+=d.getName()+" ";
  }
  str+="\n";
  return str;
 }

}
```

```
import java.awt.*;
import java.applet.*;
import java.net.*;
import java.util.*;

public class Navigator extends Frame {
  /* Overview: This class is the GUI for the applet. It is a frame with a 3 by 3
     grid of images. Each image cell represents materials in the Tangent system.
     The class allows the user to select a material which will cause the material's
     URL to be displayed in the web browser. The GUI will then be updated to
     reflect the user's choice and past history
   */


 private Tangent engine;
 private Image offScreen;
 private UpdateThread[] updateThreads;
 private Hashtable imageFetchers;
 private UpdateThread curUT;
 private Font myFont, myFont2;
 private Applet parent;

 // overide update to eliminate flicker
 public void update(Graphics g){
 }

 public void cleanup(){
  /* requires:
     modifies: this
     effects:  cleans up this. Stops update threads and any image fetcher threads
               still running
   */
  System.out.println("dispose called");
  int len = updateThreads.length;
  for(int i=0; i<len; i++)
   updateThreads[i].stop();
  Enumeration e = imageFetchers.elements();
  while(e.hasMoreElements()){
   ImageFetcher imgF = (ImageFetcher) e.nextElement();
   imgF.stop();
  }
  hide();
  super.dispose();
 }

 public Navigator(Tangent engine, Applet parent){
  super("Tangent Navigator");
  this.parent = parent;
  setResizable(false);
  setSize(300, 400);
  myFont = new Font("TimesRoman", Font.BOLD, 14);
  myFont2 = new Font("TimesRoman", Font.PLAIN, 12);
  setBackground(Color.black);
  this.engine = engine;

  // initialize image fetchers
  imageFetchers = new Hashtable();
  Enumeration e = engine.getMaterials();
  System.out.println("materials fetched from engine");
  while(e.hasMoreElements()){
   Material m = (Material) e.nextElement();
   ImageFetcher imgF = new ImageFetcher(m, this);
   imageFetchers.put(m.getName(), imgF);
   // start pre-fetching
   imgF.start();
  }

  System.out.println("image fetchers spawned");

  // initialize update threads
  updateThreads = new UpdateThread[9];
  Enumeration materials = engine.getMaterials();
  Vector curPicts;
  int counter = 0;
  for(int y=0; y<3; y++){
   for(int x=0; x<3; x++){
    // requires materials has at least 9 elements
    Material m = (Material) materials.nextElement();
    ImageFetcher imgF = (ImageFetcher) imageFetchers.get(m.getName());
    updateThreads[counter] = new UpdateThread(this, imgF, x*100, y*100+100, 1(
    updateThreads[counter].start();
    counter++;
   }
  }
  System.out.println("update threads spawned");
  curUT = updateThreads[0];
  show();
 }
```

```
public void paint(Graphics g){
  /* requires: g is not null
     modifies:
     effects:  paints the Go to URL in white
     */
  //offScreen = createImage(400, 400);
  //Graphics g2 = offScreen.getGraphics();
  g.setColor(Color.white);
  g.setFont(myFont);
  g.drawString("Go to URL: ", 30, 80);
}


private void updateChoices(){
  /* requires: engine is not null
     modifies: engine, update threads
     effects:  gets the 3 most active materials, 3 most interested materials, and
               3 random materials from engine. Sets these 9 to be displayed by
               getting their image fetcher threads and passing them to update
               threads.
     */

  Vector choices = engine.getMaterialChoices(3, 3);
  int counter = 0;
  Material m;
  while(counter<3){
    m = engine.getRandomMaterial();
    if(!choices.contains(m)){
      choices.addElement(m);
      counter++;
    }
  }

  int r1, r2, r3, r4, r5, r6;
  r1 = (int) (Math.random() * 8);
  r2 = (int) (Math.random() * 8);
  r3 = (int) (Math.random() * 8);
  r4 = (int) (Math.random() * 8);
  r5 = (int) (Math.random() * 8);
  r6 = (int) (Math.random() * 8);
  // randomize each material's location in the GUI
  Material m2;
  m = (Material) choices.elementAt(r1);
  m2 = (Material) choices.elementAt(r2);
  choices.setElementAt(m, r2);
  choices.setElementAt(m2, r1);
  m = (Material) choices.elementAt(r3);
  m2 = (Material) choices.elementAt(r4);
  choices.setElementAt(m, r4);
  choices.setElementAt(m2, r3);
  m = (Material) choices.elementAt(r5);
  m2 = (Material) choices.elementAt(r6);
  choices.setElementAt(m, r6);
  choices.setElementAt(m2, r5);


  for(int i =0; i<9; i++){
    m = (Material) choices.elementAt(i);
    ImageFetcher imgF = (ImageFetcher) imageFetchers.get(m.getName());
    updateThreads[i].setFetcher(imgF);
  }
  System.out.println("choices updated");
}

// java 1.0 even model for maximum compatibility
public boolean mouseDown(Event e, int x, int y){
  /* requires: e is not null
     modifies: engine, updateThreads
     effects:  triggers the material, if any, located at (x, y), then redraws
               a black square where the material images used to be. Shows
               the selected material's URL in a HTML frame
     */

  if(y>100){
    UpdateThread ut = getUTAt(x, y);
    Material m = ut.getFetcher().getMaterial();
    engine.triggerMaterial(m.getName());
    updateChoices();
    Graphics g = getGraphics();
    g.fillRect(0, 100, 300, 300);
    AppletContext ac = parent.getAppletContext();
    ac.showDocument(m.getURL(), "main");
  }
    return true;
}
```

```
public synchronized boolean mouseMove(Event e, int x, int y){
 /* requires: e is not null
    modifies: updateThreads
    effects:  displays the URL of the material the mouse is over, sets that
              material to flicker in the display
    */
 if(y>100){
  UpdateThread ut = getUTAt(x,y);
  if(ut==curUT)
   return true;
  // correct the speed of the square the cursor used to be in
  curUT.setDelay(1000);
  //curUT.deactivate();
  //curUT.explicitDraw();
  // speed up the display of the square the cursor is in
  System.out.println("mat name: "+ut.getFetcher().getMaterial().getName());
  curUT = ut;
  curUT = getUTAt(x, y);
  curUT.setDelay(100);
  Graphics g = getGraphics();
  g.setColor(Color.black);
  g.fillRect(100, 0, 200, 100);
  g.setFont(myFont2);
  g.setColor(Color.yellow);
  g.drawString(curUT.getFetcher().getMaterial().getURL().toExternalForm(), 110, 80);
 }
 return true;
}

public boolean handleEvent(Event e){
 System.out.println("handle Event");

 switch(e.id){
 case Event.MOUSE_DOWN:
  return mouseDown(e, e.x, e.y);
 case Event.MOUSE_MOVE:
  return mouseMove(e, e.x, e.y);
 case Event.WINDOW_DESTROY:
  System.out.println("window destroy");
  cleanup();
  break;
 }
 return true;
}

private UpdateThread getUTAt(int x, int y){
 /* requires: x>0, y>0
    modifies:
    effects: returns the updateThread corresponding to the image at (x,y). If
             no such thread exists, returns the thread at (0,0)
    */
 int xDiv = x / 100;
 int yDiv = (y-100) / 100;

 switch (yDiv) {
 case 0:
  switch(xDiv) {
  case 0:
   return updateThreads[0];
  case 1:
   return updateThreads[1];
  case 2:
   return updateThreads[2];
  }
 case 1:
  switch(xDiv) {
  case 0:
   return updateThreads[3];
  case 1:
   return updateThreads[4];
  case 2:
   return updateThreads[5];
  }
 case 2:
  switch(xDiv) {
  case 0:
   return updateThreads[6];
  case 1:
   return updateThreads[7];
  case 2:
   return updateThreads[8];
  }
 default:
  return updateThreads[0];
 }
}
```

```
}
```

```
import java.util.*;


public class Tangent {
 /* Overview: This class can take a manage a set of Materials that each have
              an associated set of descriptors. As a materials are selected,
              the system can be queried as to which materials it thinks are
              most relevant to the user's interests based on past history.
  */


 private Hashtable descriptors;
 private Hashtable materials;
 private Vector activeDescriptorSet;
 private Vector interestDescriptorSet;
 // determines how many hits a descriptor requires before it is set to interest
 private int interestThreshold;


 public Tangent(int interestThresh){
  descriptors = new Hashtable(20);
  materials = new Hashtable(30);
  activeDescriptorSet = new Vector(3);
  interestDescriptorSet = new Vector(10);
  interestThreshold = interestThresh;
 }



 public void addMaterial(Material newMaterial) throws TangentException {
  /* requires: descriptors, materials
     modifies:
     effects: adds newMaterial to the system.
     */
  String key = newMaterial.getName();
  if(materials.get(key) != null)
   throw new TangentException("Tangent.addMaterial: "+newMaterial.getName()+" already exists");
  else {
   // add the material
   materials.put(key, newMaterial);
   Enumeration e = newMaterial.getDescriptors().elements();
   // add the descriptors that describe it
   while(e.hasMoreElements()){
    Descriptor d = (Descriptor) e.nextElement();
    key = d.getName();
    // only add new descriptors
    if(descriptors.get(key) != null)
     descriptors.put(key, d);
   }
  }
 }


 public boolean triggerMaterial(String materialName){
  /* requires:
     modifies:
     effects: returns false if the material named materialName does not exist
     */
  Material m = (Material) materials.get(materialName);
  if(m == null)
   // material does not exist
   return false;

  // run calculations to adjust Tangent state
  activeDescriptorSet = m.getDescriptors();
  Descriptor d;
  Enumeration e;
  // fade each descriptor
  e = descriptors.elements();
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   d.fade();
  }
  // hit each active descriptor
  e = activeDescriptorSet.elements();
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   d.hit();
   // hit a second time to compensate for the previous fade (optimization)
   d.hit();
   // find if any of the descriptors are a person's interest
   if(d.numHits() ≥ interestThreshold){
    if(!d.isInterested()){
     d.makeInterested();
     if(!interestDescriptorSet.contains(d))
      interestDescriptorSet.addElement(d);
    }
   }
```

```
  }

  // Material triggered
  return true;
}

public Material getRandomMaterial(){
  /* requires: materials is not null
     modifies:
     effects: returns an random material stored in this system
     */
  int total = materials.size();
  double mat = Math.random() * (double) total;
  int index = (int) mat;

  System.out.println("index: "+index);

  Enumeration e = materials.elements();
  Material m;

  m = (Material) e.nextElement(); // to get around jdk compile warning/error
  for(int i=1; i< index; i++)
    m = (Material) e.nextElement();

  return m;
}

public void treeTraverse(BTree node, int numNodes, Vector nodes){
  /* requires: node is not null, nodes is not null, numNodes ≥0
     modifies: node, numNodes, nodes
     effects: retrieves numNodes amount of the greatest elements in node (tree)
     */
  if(numNodes == 0)
    return;
  Material m;
  if(node.getLeft() == null || node.getLeft().wasTraversed()){
    if(node.getRight() == null || node.getRight().wasTraversed()){
      // either at a leaf node or a node that has already been added to nodes
      if(!node.wasTraversed()){
        m = node.getMaterial();
        // only add the material if it's not already in nodes
        node.makeTraversed();
        if(!nodes.contains(m)){
          nodes.addElement(m);
          treeTraverse(node.getParent(), numNodes-1, nodes);
        }else
          // no right branch or right already added in nodes
          treeTraverse(node.getParent(), numNodes, nodes);
      }else
        treeTraverse(node.getParent(), numNodes, nodes);
    }else{
    // no left branch or left already added in nodes right side can be searched
      if(node.getRight().wasTraversed())
        treeTraverse(node.getParent(), numNodes, nodes);
      else{
        m = node.getMaterial();
        node.makeTraversed();
        if(!nodes.contains(m)){
          nodes.addElement(m);
          treeTraverse(node.getRight(), numNodes-1, nodes);
        }else
          treeTraverse(node.getRight(), numNodes, nodes);
      }
    }
  }
  else
    treeTraverse(node.getLeft(), numNodes, nodes);
}


public Vector getMaterialChoices(int numActive, int numInterests){
  /* requires: materials contains at least one element
     modifies:
     effects: returns numActive amount of materials that are most active (based
              on current active material's descriptors) and numInterest
              amount of materials that are most active (based on past history)
   */

  // calculate the long representation of the user's interests
  Vector choices = new Vector(numActive+numInterests);
  long activeDescriptorVal=0;
  long interestDescriptorVal=0;
  Descriptor d;

  Enumeration e = activeDescriptorSet.elements();
  while(e.hasMoreElements()){
    d = (Descriptor) e.nextElement();
    activeDescriptorVal+=d.getID();
```

```
  }

  e = interestDescriptorSet.elements();
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   interestDescriptorVal+=d.getID();
  }


  // fill choices with materials corresponding to active descriptor and
  // interest descriptor sets

  // put each material in a binary tree according to the descriptor
  // intersection

  e = materials.elements();
  Material m;
  m = (Material) e.nextElement();
  BTree activeTree = new BTree(m, intersection(activeDescriptorVal, m.getDescriptorIDs()));
  BTree interestTree = new BTree(m, intersection(interestDescriptorVal, m.getDescriptorIDs()));

  while(e.hasMoreElements()){
   m = (Material) e.nextElement();
   int newActiveVal = intersection(activeDescriptorVal, m.getDescriptorIDs());
   int newInterestVal = intersection(interestDescriptorVal, m.getDescriptorIDs());

   activeTree.addNode(m, newActiveVal);
   interestTree.addNode(m, newInterestVal);
  }
  System.out.println("trees created");

  // traverse the tree to retrieve the most relavent nodes
  treeTraverse(interestTree, numInterests, choices);
  treeTraverse(activeTree, numActive, choices);
  System.out.println("choices len: "+choices.size());
  return choices;
 }

 private int intersection(long interests, long descriptorSet){
  /* requires: interests and descriptorSet >0
     modifies:
     effects: returns a long representation of an intersection between the
              sets represented by interests and descriptorSet
     */
  long intersect = interests & descriptorSet;
  int hits = 0;
  int bit = 1;
  while(bit < 65){
   if((intersect & 1)==1)
    hits++;
   intersect>>=1;
   bit++;
  }
  return hits;
 }


 public String toString(){
  /* requires:
     modifies:
     effects: returns a string representation of this
     */
  String str = new String();
  Enumeration e = descriptors.elements();
  str+="<--- Descriptors --->\n\n";
  Descriptor d;
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   str+=d.getName()+"\n";
  }
  str+="\n<--- Materials --->\n";
  Material m;
  e = materials.elements();
  while(e.hasMoreElements()){
   m = (Material) e.nextElement();
   str+=m.toString()+"\n";
  }
  str+="\n<--- Active Descriptor Set --->\n";
  e = activeDescriptorSet.elements();
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   str+=d.getName()+"\n";
  }
  str+="\n<--- Interest Descriptor Set --->\n";
  e = interestDescriptorSet.elements();
  while(e.hasMoreElements()){
   d = (Descriptor) e.nextElement();
   str+=d.getName()+"\n";
```

```
    }
    str+="\n";
    return str;
  }

  public Enumeration getMaterials(){
    /* requires:
       modifies:
       effects: returns an enumeration of all materials stored in this
       */
    return materials.elements();
  }
}
```

```
public class TangentException extends Exception {

  public TangentException(){
    super();
  }

  public TangentException(String msg){
    super(msg);
  }
}
```

```
import java.util.*;
import java.net.*;


public class TestApplet {

  public static void main(String args[]){
    Tangent engine = new Tangent(1);
    System.out.println("tangent instantiated");

    Material m1, m2, m3, m4, m5, m6, m7, m8, m9;

    Descriptor d1, d2, d3;

    d1 = new Descriptor("d1", 1);
    d2 = new Descriptor("d2", 2);
    d3 = new Descriptor("d3", 4);

    Vector dVec1 = new Vector();
    Vector dVec2 = new Vector();

    dVec1.addElement(d1);
    dVec1.addElement(d2);
    dVec1.addElement(d3);

    dVec2.addElement(d2);
    dVec2.addElement(d3);

    Vector pVec1 = new Vector();
    Vector pVec2 = new Vector();

    String eye = new String("http://deepc.mit.edu/eye.gif");
    String tubby = new String("http://deepc.mit.edu/tubby.jpg");
    String elway = new String("http://deepc.mit.edu/elway.jpg");

    pVec1.addElement(eye);
    pVec1.addElement(tubby);
    pVec1.addElement(elway);

    pVec2.addElement(tubby);
    pVec2.addElement(elway);
    try{
      m1 = new Material("m1", "http://deepc.mit.edu", dVec1, pVec1);
      m2 = new Material("m2", "http://deepc.mit.edu", dVec2, pVec2);
      m3 = new Material("m3", "http://deepc.mit.edu", dVec1, pVec1);
      m4 = new Material("m4", "http://deepc.mit.edu", dVec1, pVec1);
      m5 = new Material("m5", "http://deepc.mit.edu", dVec2, pVec2);
      m6 = new Material("m6", "http://deepc.mit.edu", dVec1, pVec1);
      m7 = new Material("m7", "http://deepc.mit.edu", dVec1, pVec1);
      m8 = new Material("m8", "http://deepc.mit.edu", dVec1, pVec1);
      m9 = new Material("m9", "http://deepc.mit.edu", dVec2, pVec2);

      engine.addMaterial(m1);
      engine.addMaterial(m2);
      engine.addMaterial(m3);
      engine.addMaterial(m4);
      engine.addMaterial(m5);
      engine.addMaterial(m6);
      engine.addMaterial(m7);
      engine.addMaterial(m8);
      engine.addMaterial(m9);
      System.out.println("materials added");
    }
    catch(TangentException e){
      System.out.println("tangent exception: "+e.getMessage());
    }

    catch(MalformedURLException e){
      System.out.println("malformed url: "+e.getMessage());
    }

    Navigator n = new Navigator(engine);
    System.out.println("navigator instantiated");
  }


}
```

```
import java.util.*;
import java.net.*;
import java.io.*;


public class TestApplet2 {

 public static void main(String args[]){
  try{
   Tangent engine = new Tangent(1);
   URL url = new URL("http://deepc.mit.edu/Peter/AUP/initFiles/init2");
   InputStream  is = url.openStream();
   System.out.println("stream opened");
   InitTangent creator = new InitTangent(is, engine);
   creator.init();
   System.out.println("tangent intialized");
   Navigator n = new Navigator(engine);
   System.out.println("navigator instantiated");
  }

  catch(MalformedURLException e){
   System.out.println("malformed url: "+e.getMessage());
  }
  catch(IOException e){
   System.out.println("io: "+e.getMessage());
  }
 }


}
```

```
import java.net.*;
import java.io.*;


public class TestTangent {

 private static Tangent engine;

 public static void main(String args[]){

  // interest threshold
  // some of these should be moved to the navigator
  engine = new Tangent(1);

  try{
   URL url = new URL("http://deepc.mit.edu/Peter/AUP/initFiles/tangent.init");
   InputStream  is = url.openStream();
   System.out.println("stream opened");
   InitTangent creator = new InitTangent(is, engine);
   creator.init();
   System.out.println("tangent intialized");

  }
  catch(MalformedURLException e){
   System.out.println("malformed url: "+e.getMessage());
  }
  catch(IOException e){
   System.out.println("IO: "+e.getMessage());
  }

  //System.out.println(engine.toString());
  //sleep(3000);
  testTrigger();
  //System.out.println(engine.toString());

  testGetMaterialChoices();
  testRandom();
 }

 public static void testRandom(){
  for(int i=0; i<40; i++)
   System.out.println("rand mat: "+engine.getRandomMaterial().getName());

 }

 public static void testGetMaterialChoices(){
  engine.triggerMaterial("m1");
  engine.getMaterialChoices(1,2);
 }

 public static void testTrigger(){
  engine.triggerMaterial("m1");
  engine.triggerMaterial("m2");
  engine.triggerMaterial("m4");
  //engine.triggerMaterial(m3);
 }

 public static void sleep(long t){
  try{Thread.sleep(t);}
  catch(InterruptedException e){}
 }


}
```

```
import java.util.*;
import java.awt.*;
import java.lang.Thread;
import java.applet.*;

public class UpdateThread extends Thread {
 /* Overview: this thread is responsible for updating the image in a cell on
              the GUI. It has a helper image fetcher threads that takes care
              of the images it is responsible for painting on the GUI

 */

 private ImageFetcher curFetcher;
 private int x, y, height, width;
 private Navigator gui;
 private Graphics g;
 private long delay;
 private int len = 0;

 public UpdateThread(Navigator parent, ImageFetcher fetcher, int x, int y, int height, int width){
  curFetcher = fetcher;
  gui = parent;
  this.x = x;
  this.y = y;
  this.height = height;
  this.width = width;
  delay = 1000;
 }


 public void run(){
  /* requires:  curFetcher, gui are not null.  x,y,height,width,delay ≥0
     modifies:
     effects:  squentially paints all images that curFetcher has completely loaded
               waits delay amount of time before painting the next image
     */
  len = curFetcher.numImgs();
  Image img;

  while(true){

   for(int i = 0; i<len; i++){
    synchronized(curFetcher){
     if(curFetcher.isLoaded(i)){
      img = curFetcher.getImage(i);
      draw(img);
     }
    }
    try{
     sleep(delay);
    }
    catch(InterruptedException e){
     System.out.println("interrupted: "+e.getMessage());
    }
   }
  }
 }

 public void draw(Image img){
  /* requires: img is not null
     modifies:
     effects:  paints img to the GUI in this' cell
     */
  g = gui.getGraphics();
  g.drawImage(img, x, y, width, height, gui);
  g.dispose();
 }

 public synchronized void setDelay(long newDelay){
  /* requires:
     modifies: delay
     effects:  sets the wait time in between image updates to newDelay mill. secs.
     */
  delay = newDelay;
 }

 public synchronized void setFetcher(ImageFetcher newFetcher){
  /* requires:
     modifies:  curFetcher
     effects:   sets this' image fetcher to newFetcher, thus changing the images
                to be painted
     */
  curFetcher = newFetcher;
  len = newFetcher.numImgs();
 }

 public synchronized ImageFetcher getFetcher(){
  /* requires:
```

```
    modifies:
    effects: returns this' image fetcher
    */
  return curFetcher;
 }
}


/*public void explicitDraw(){
  Image img = curFetcher.getImage(0);
  draw(img);
 }*/
```