Interactive Web Shopping: Building Trust through Storytelling and Interface Cues

by

Jonathan E. Shoemaker

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 23, 1997

Author [illegible] ____________________
Department of Electrical Engineering and Computer Science
May 23, 1997

Certified by [illegible] ____________________
Glen L. Urban
Dean, MIT Sloan School of Management, Thesis Supervisor

Accepted by [illegible] ____________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Interactive Web Shopping: Building Trust through Storytelling and Interface Cues

by

Jonathan E. Shoemaker

May 23, 1997

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Companies are becoming increasingly excited about the marketing potential of the Web. Consumers, however, remain wary about the potential problems with conducting transactions over the Internet and view retail Web sites with suspicion. Many of the current sites lack interface cues that could build trust with consumers and entertain them while they shop. In the following thesis, I discuss the concept of trust and how to improve current sites using trust building cues and interactive interfaces. I also describe the development and implementation of the Java software framework that I designed for a prototype site and its potential for expansion. Lastly, I explore the possibilities of adding role playing scenerios to retail Web sites and introduce further topics for research.

Thesis Supervisor: Glen L. Urban
Title: Dean, MIT Sloan School of Management

Acknowledgments

I dedicate this thesis to my family, whose financial support allowed me to come to MIT and whose emotional support helped me to survive the past five years. My parents, Harold and Sandy Shoemaker, taught me the importance of hard work and always applauded my efforts, despite the outcome. I would like to thank my sister, Susan Shoemaker, for always listening to my problems and giving good advice.

I would also like to thank MIT's *La Maison Française* for five years of friendship, good food, and laughter. It has always been my "home" at MIT, and I will miss it in the years to come.

I am also extremely grateful for Jeffry Kahle, my best friend, for his emotional support. For nearly three years he has managed to keep me sane, despite my engineering studies and workload.

Lastly, I would like to thank my associates at the MIT Sloan School of Management. I greatly appreciate the guidance and assistance of Dr. William Qualls and Dr. Glen Urban, with whom I have had the honor of working during the past year. Ahmed Benabadji and Frank Days have also been valuable to me while I worked on the project. I thank them for their friendship and moral support.

Contents

List of Figures

Introduction

Web Marketing Opportunities

A report by Alba et al. (1996) predicts that Web-based sales will sum from 5 to 300 billion dollars by the year 2000. Many retailers, hearing these predictions, are rushing to the Web to capture a chunk of the expanding market of middle- to upper-income Internet users. Their target consumers typically have high disposable incomes and have little time to spend comparison shopping.

Shortcomings of Current Retail Web Sites

The methodology and interface of most retail Web sites face some serious drawbacks. Instead of taking advantage of the potential that the Internet offers, in terms of interactivity and information collection, most sites simply provide online electronic catalogs. Often, these catalogs only include a short description of the item, a price, and an order number. If a consumer knows precisely what he wants and knows that he definitely wants to order it through a particular merchant, then the online catalog works well.

Unfortunately, for some goods, a consumer has little or no knowledge about what he really wants. Online catalogs do little to help him decide to buy a particular good since he may have no idea what product by any single manufacturer will best serve his needs while fitting within his budget. For example, a first-time car buyer may not have brand preferences and may be looking for a specific type of vehicle (e.g., sports utility), but knows nothing about the available products on the market. In this case, he would probably want to talk to his friends, family, or a knowledgeable salesperson (Alba, 1996). He would probably not want to visit the Web sites of ten to fifteen different auto manufacturers, collect all the statistics of the available models, and decide from those what to buy.

In addition, the consumer must trust that the Web site provides him with accurate information and can deliver on its promises. Many current retail sites give only a phone number to call for placing an order or for asking questions. Sometimes, when the consumer does call to order a product, he finds that its real price is not the one quoted on the site. Furthermore, if he sends his credit card information over the Internet to purchase the item, he must trust that he will be charged the stated price and that his credit card information will not be misued. Due to these issues, consumers remain wary about purchasing products over the Web.

Suggestions for Improvements

The model retail Web site should build consumer trust, especially if it attempts to sell high-priced durable goods, such as automobiles. Trust-building should be its first priority, since consumers will be more likely to purchase from the site if they know they can trust it. Depending on the products the site tries to sell, the effort required to gain consumer trust could be substantial.

The model site should also provide a service to the consumer. It should offer assistance and guidance, and it should be fun and easy to use. Instead of hiding long product and price lists under layers of nested Web pages (as many sites do to discourage comparison shopping), a site should smoothly guide the consumer through the shopping process, particularly if he has no idea what he wants to buy (Alba, 1996). Conversely, if the consumer knows what he wants, it should also provide easy access to information about the product's price and availability.

The whole entire selection process, therefore, must be quick and relatively painless, yet also thorough and trustworthy enough that a buyer feels he is sufficiently maximizing the utility he will gain through the purchase. Some recent studies have shown that computer users typically spend less than ten minutes at a particular site unless it is particularly entertaining or interactive. Thus, except for shoppers who have already decided on a particular purchase, any site which nests long lists of items and hides prices without an interactive search or selection mechanism likely loses market share of undecided or casual shoppers.

Examining these factors, the main research question is this: How may one best design and implement interactive, Web-based marketing systems which are at once fun, trustworthy, easy to use, and fast enough to keep the interest of casual shoppers and Web surfers?

Recent Research

In late 1996, Glen Urban, Dean of the Sloan School of Management, proposed a project to study the usage and effects of an interactive site including the ideal elements described above. The prototype site would allow the user to shop for a pickup truck from any manufacturer several different ways, depending on the level of guidance that he required or desired. Both Ahmed Benabadji (an MBA student at Sloan) and I worked together with Dean Urban on the project since December 1996, and Frank Days (another MBA student) joined the team in February 1997. Ahmed and Frank developed the site's trust-based marketing model and its general flow, while I studied the technical issues, made implementation decisions, and designed a software framework for it.

Thesis Goals and Overview

In this thesis, I address problems with current retail Web sites, introduce interface cues and tools that may build consumer trust in a site, study previous marketing multimedia projects in relation to the Web, and present a modular framework for creating an interactive retail site. While I do not imply that my framework is the only one or the best available, it can easily be extended from its present form to encompass most large projects.

Chapter Two discusses the concept of trust and relates it to online marketing. It also lists some broad trust building cues that may build consumer trust in a site. Among the cues, it presents the notion of storytelling and how it may be used to make a site more fun and attractive.

Chapter Three summarizes the marketing design of the prototype Web site that we developed. It includes a flowchart of the user's journey through the site. It also discusses the purposes for the site's four main modules and the services that they offer to the user.

Chapter Four reviews past history with multimedia marketing surveys and describes how some of their elements could be extended to the Web.

It explores the Java programming language as an option and discusses its pros and cons. Lastly, it provides evidence of the feasibility of Java for use with interface controls.

Chapter Five quickly introduces Java basics with comparisons to C++. Readers familiar with C++ (or other object-oriented languages) but unfamiliar with Java may find this short chapter helpful for understanding the terminology in Chapters Six and Seven and the code in the Appendix.

Chapter Six outlines the evolution of the software framework for the site. It provides a journal of the changes over a five-month period that led to the final prototype design. The end of the chapter briefly discusses work in progress at the time of this writing, and explains how the framework might be extended.

Chapter Seven describes the Java classes that form the software framework for the site. It does not include any code but shows the ancestral and functional hierarchies of the classes and outlines their major purposes.

Chapter Eight concludes the thesis. It reflects on the lessons I learned while working on the project and introduces ideas for future research directions. It delves into the possibilities of RPGs as it contemplates the potential of the Web within the next three to five years.

The Issue of Trust in Online Marketing

A Definition of Trust

To define trust as it pertains to Web shopping, I will refer heavily to Ahmed Benabadji's MBA thesis (Benabadji, 1997). Ahmed was a student at the MIT Sloan School while working with me on the project, and for his thesis he studied the concept of trust in the electronic marketplace. He designed a new model for how trust affects purchase decisions on the Internet and how online businesses may effectively build consumer trust.

Benabadji cited Deutsch (1958) for the following definition of trust as "an expectation of interpersonal events":

> **"An individual may be said to have trust in the occurrence of an event if he expects its occurrence and the expectation leads to behavior which he perceives to have greater negative motivational consequences if the expectation is not confirmed than positive motivational consequences if it is confirmed."**

Deutsch also suggested that trust is an issue when some future course of action is yet ambiguous, when others affect the outcome, and when negative effects of the outcome outweigh positive effects.

The Composition of Trust

To further describe some aspects of trust, Benabadji mentioned Hosmer (1997), who proposed a set of trust characteristics:

"1. Trust is generally expressed as an optimistic expectation on the part of an individual about the outcome of an event or the behavior of a person.

2. Trust generally occurs under conditions of vulnerability to the interests of the individual and dependence upon the behavior of other people.

3. Trust is generally associated with willing, not forced, cooperation and with the benefits resulting from that cooperation.

4. Trust is generally difficult to enforce.

5. Trust is generally accompanied by an assumption of an acknowledged or accepted duty to protect the rights and interests of others."

Qualities of a Trustworthy Individual

But what characterizes a trustworthy individual? What qualities make a given person more trustworthy than another? To answer those questions, Benabadji quoted five aspects from an analysis by Butler and Cantrell (1984):

"1. Integrity - the reputation for honesty and truthfulness on the part of the trusted individual.

2. Competence - the technical knowledge and interpersonal skills needed to perform the job.

3. Consistency - the reliability, predictability, and good judgment in handling situations.

4. Loyalty - benevolence, or the willingness to protect, support, and encourage others.

5. Openness - mental accessibility, or the willingness to share ideas and information freely with others."

Trust in a Software Agent

The above qualities likely help to build trust in interpersonal relationships, but what about trust between an individual and an electronic service or a software agent? Personal computers and the Web have not yet provided the technical capability for high-quality video and audio interaction between people on the Internet. Furthermore, most businesses cannot afford to staff Web sites with hundreds of people around the clock.

For the next few years, they will have to rely on software agents for most online consumer interaction. Benabadji cited Van Slyke (1996), who said that trust between a user an a software agent is determined by the following:

"1. The user's knowledge (domain and technical).
2. The predictability of the agent. (The agent performs consistently on frequent tasks.)
3. The dependability of the agent (expectation that the agent will help if needed).
4. The technical competence of the agent, or his ability to perform the tasks.
5. The 'fiduciary' responsibility of the agent.
6. The level of control given to user to monitor his agent.
7. The frequency of use."

A New Model for Trust in Online Marketing

To summarize the remainder of his work, Benabadji created a new model for trust with respect to Internet commerce because previous models proved to be inadequate. He called it the "cascade model of trust." In his view, consumers must pass a certain trust "threshold" in each stage of the shopping process before they can proceed. He cited four stages in all, including the following: trust in the channel, trust in the agent, trust in the information, and trust in the fulfillment.

The first three stages occur before the consumer decides to purchase a product, and the last occurs afterwards, primarily affecting merchandise returns and repeat purchases. The first stage, trust in the channel, implies that the consumer must trust the method by which he shops for a product. When shopping over the Internet, the consumer wants assurance that his transactions are safe and that his private information remains private. The second stage, trust in the agent, suggests that the consumer must believe that the software agent (or live salesperson) wants only to serve his needs as well as possible.

The third stage, trust in the information, progresses when the consumer accepts the credibility of the agent and believes the information he receives. The consumer reaches the fourth and last stage in the process after he has decided to make the purchase and relies on the agent to honor his promises, deliver the product, and guarantee its worth. How the agent performs during the last step affects whether the consumer will return to purchase again.

Applying the Model to Web Content

Relating the cascade model to Web site content, Benabadji promoted trust building cues, or "signals in the environment that result in a higher perception of trustworthiness by the consumer." Through focus group studies and surveys, he concluded that the five following dimensions of trust were the most important to online shoppers:

"1. 'I trust what I understand.' Consumers want predictability and consistency from an agent. They also want to be able to assess its relative competence in providing recommendations.

2. 'I trust processes and information that I control.' The consumer should be able to roam the site freely, feeling that the site caters to his needs. He wants to see only what he cares about, and does not want to have a lot of useless information or advertising "pushed" at him.

3. 'I trust Web sites that provide positive and negative information.' Consumers want honesty, integrity, and openness.

4. 'I trust people.' The site must emphasize likeability and friendliness, just like many successful salespeople.

5. 'I trust independence.' Consumers value benevolence and

> **dependability when assessing the trustworthiness of an information provider."**

The fifth point, "I trust independence," was repeated many times by the focus groups, who referred to *Consumer Reports* and the Edmunds Web site. They praised those sources for not taking money from manufacturers or advertisers and for being open and honest about prices, features, specifications, and quality.

Suggestions for Site Improvements

Using Benabadji's five major dimensions of trust, I will attempt to suggest improvements that Web sites can implement to build consumer trust throughout the three pre-purchase shopping steps. The first step, trust in the channel, implies that the consumer has to believe that the Internet will protect his privacy, including his name, address, and credit card information. To build this trust, a site could show that it provides complete encryption of transactions (as the Dell Computer site currently does) and promise that *no* information would be given to *anyone* not involved with a particular transaction. A company could back up these statements with its own address and contact information (including a toll-free number) for verification and for legal purposes should it break its promises.

Building trust past the second threshold can be very difficult for Web sites, since the agent is primarily text and images on a computer screen. Instead of interpreting facial and vocal cues from a live salesperson, a consumer must rely on only what he sees (or might hear) on the site. He must be able to control his experience, instead of the site controlling it for him. Just as when a shopper tells a pushy salesperson "I'm only looking," in order to be left alone, the consumer should be able to avoid an agent if he so desires.

Furthermore, to emulate successful salespeople, the interface of the site must be friendly and interactive. The graphics should be pleasing and comforting, yet brilliant and exciting. Background images that provide a nice backdrop or give some reference point for the site can be very effective. Hsu (1996) suggests that blue objects reduce eye strain, and focus group results from Benabadji (1997) imply that blue is a very "trustworthy" color. In addition, all controls should be intuitive and easy to use. Users hate bulky, non-descript controls that never respond or react after a long delay. The design of interactive controls should not be so ambitious that a modern computer cannot quickly draw them or respond to them.

As sites progress, their agents or interfaces can include "storytelling" elements. As salespeople tell stories and give personal accounts to familiarize themselves with customers, so could a Web site. It could provide a backdrop, or environment in which the consumer feels confortable as soon as he enters. The consumer should be able to talk to any agent to which he feels he might relate and to shop freely with its help. The agent, of course, may not have to follow the consumer on every step of his "journey" through the site, but he can "drop by" from time to time and should return quickly when needed. As the technology progresses, sites may even provide animated agents in the absence of real individuals. Chapter Eight of this thesis discusses possible spin-offs from the storytelling idea.

Building trust to the third threshold is perhaps conceptually easier to do, but many sites miss some of the subtleties. If a site provides a service, and does not sell but suggests products, it should remain independent from all manufacturers and provide non-biased reviews. Consumers expect to find both positive *and* negative reviews; if they do not see any negative comments, they will lose confidence in the information source. Since communication within newsgroups often represents contrasting opinions, a site could support threaded discussions among its users to provide views from all sides. Furthermore, if a manufacturer wants to build trust in the information it presents on its site, it should also give its own positive and negative reviews of its products, and honestly show how each rates to the competition on *all* valid points. Obviously, most manufacturers can create unique lists of attributes that set their products apart from all others. Consumers sometimes see past this particular ploy and lose confidence.

Consumers greatly respect sites, however, that have undergone third-party verification and have received rave reviews. Successful sites are just now beginning to proclaim their awards from trade magazines and evaluation firms. They also have reciprocal links to other well-known and trusted sites to assure their users that they will provide legitimate services. Sites that stand alone and reference no other sites seem to falter unless they have already secured a large following through other arenas, such as print media or television.

Introduction of Our Prototype Site

Considering these cues, our research team designed a prototype site that would build consumer trust. We chose to study automobile shopping because automobiles were expensive durable goods and their salespeople were considered to be notoriously untrustworthy (Benabadji, 1997). We limited the vehicle selection to pickup trucks to reduce complexity. The following chapter presents the marketing design and layout of the site and its main modules.

The Overall Plan for the Prototype

Following the Cascade Model of Trust

The marketing model for our prototype followed Benabadji's cascade model of trust, and implemented some of the ideas mentioned in the last chapter. (The Appendix contains a list of some specific trust building cues we considered for the prototype and for future versions of the site.) While we would have liked to have used all of them, we could not fully develop them all in time for the June 1997 prototype, nor did we have sufficient evidence supporting the relative effectiveness of the ideas.

The Four Main Modules

As shown in the figure below from Benabadji (1997), the site contained the following four modules: expert advisor's questions and an-

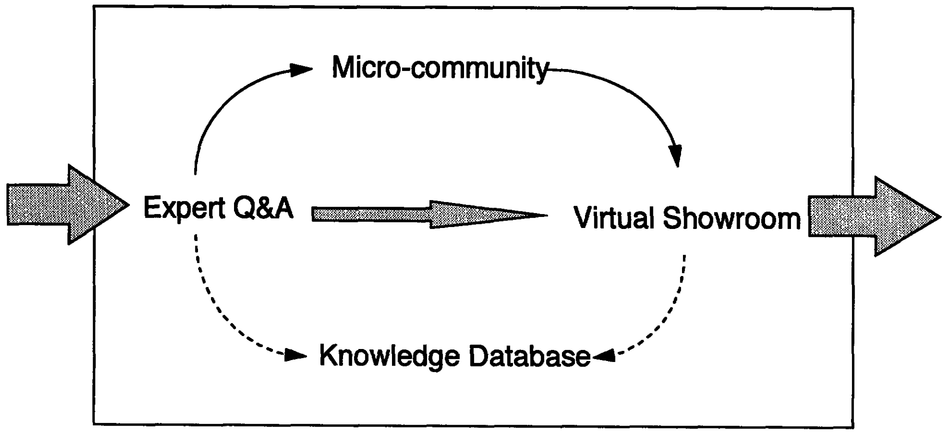


Figure 3.1: The prototype's four main modules

swers, microcommunity, virtual showroom, and knowledge database. At any time, the user could choose to visit any one of the four modules, although we preferred that the user take advantage of the advisor first. At the onset of the expert advisor module, we allowed the user to choose an advisor from a panel of "experts."

Once the user had chosen his advisor, the advisor would present him with a series of questions about his preferences and planned usage, as shown in Figure 3.2, courtesy of Frank Days. After the user had answered all of the questions, the advisor would calculate best-fit matches between the response data and the available trucks and it would present the results. (Days and Dean Urban together formulated the flow of the questions and developed the matching algorithm.) The user could then take the results and enter the virtual showroom (or virtual "auto show") to look at the selected vehicles. If he did not yet want to see them, he could visit the other modules instead.

The prototype virtual showroom module contained specifications and pictures for a small subset of vehicles. In future versions, however, the virtual showroom could include full three-dimensional models, similar to or better than those currently shown on sites like Honda's or BMW's. The

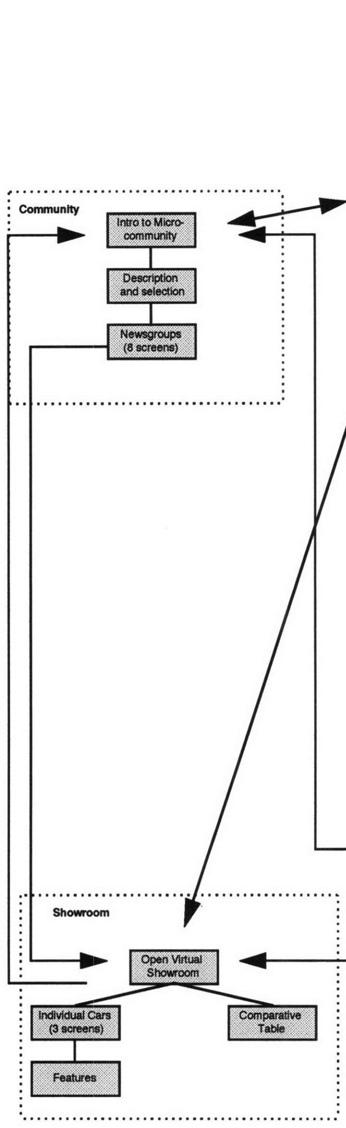


Figure 3.2: An overall flowchart for the prototype site

user could look at the vehicles, check out third-party comments, and get pricing details. In addition, the user could even order a vehicle directly from online dealers who promised to honor the prices shown.

The microcommunity module in the prototype asked the user some additional questions about his vehicle preferences so that it could determine a market segment in which to place him. It would then assign him to a particular newsgroup corresponding to his segment. If he did not like the group the module chose, it would also allow him to choose another group from the entire list. Once in a group, the user could read comments from others or post his own. Our site would not censor the remarks, so readers could find both positive and negative comments. We hoped to build trust in the site through this cue of contrasting reviews, as suggested by Benabadji (1997).

Lastly, the knowledge database would provide the user with any information he desired about a particular vehicle. We decided that the easiest way to implement it for early versions would be as a hierarchy of HTML pages. Later versions could add a search engine to simplify the lookup of an obscure item, and could offer movie clips describing the operation and maintenance of various vehicle components.

Remaining Topics

With a month remaining at the time of this writing before the demonstration of the prototype, I hesitate to include further details which could later be incorrect. The rest of this thesis discusses my programming background, the evolution of my software framework for the site, and the framework's final implementation.

Marketing and My Software Experience

Marketing and Its Interest in Computers

The idea of using a trustworthy, user-friendly interface for marketing purposes is certainly not new. Marketing analysts have already been using computers for quite a few years to collect consumer preference data about new and existing products and services. Due to the ever increasing cost of developing and producing new products, especially high-tech durable goods, many companies want to know before they commit to a large investment what features consumers want most and how much they might be willing to pay.

"Information Acceleration" Multimedia Surveys

One approach, described by Urban et al. (1996) as information acceleration (IA), uses a multimedia survey to present a potential product to a user as if it already exists. The interactive survey gives the user some information about the product and its concept and then allows him to peruse some advertising, a few mock news releases, and even several "word-of-mouth" interviews, in which current owners (or subscribers to a service) talk about the aspects they like or dislike. The respondent must then answer a series of questions about the product, particularly his preferences for some of its features, how he favors it versus other products in its class, and how much he might be willing to pay. While the advertisements, news releases, and interviews are often created specifically for the surveys, they are not intended to sway the respondent toward buying the product but rather to gain valuable information about what may yet be changed about the product to make it more desirable to a greater chunk of the market.

A couple of these surveys have been produced for GM (Urban, 1996), and for the French electric utility Electricité de France (EDF). Typical interfaces for these programs include visual controls for preference indication, much like the example shown in Figure 4.1 on the next page. To distribute points among the items, the respondent may use the arrows (small arrows for allocating one point at a time, large for ten) or he may click directly on the scales. The current levels of the scales are represented by red arrows on the scales and numbers in the boxes beneath the scales (not shown in the figure). This particular interface shows a spatial relationship of the preference allocation through the arrows in addition to the abstract, quantitative relationship given by the numbers at the bottom of the screen.

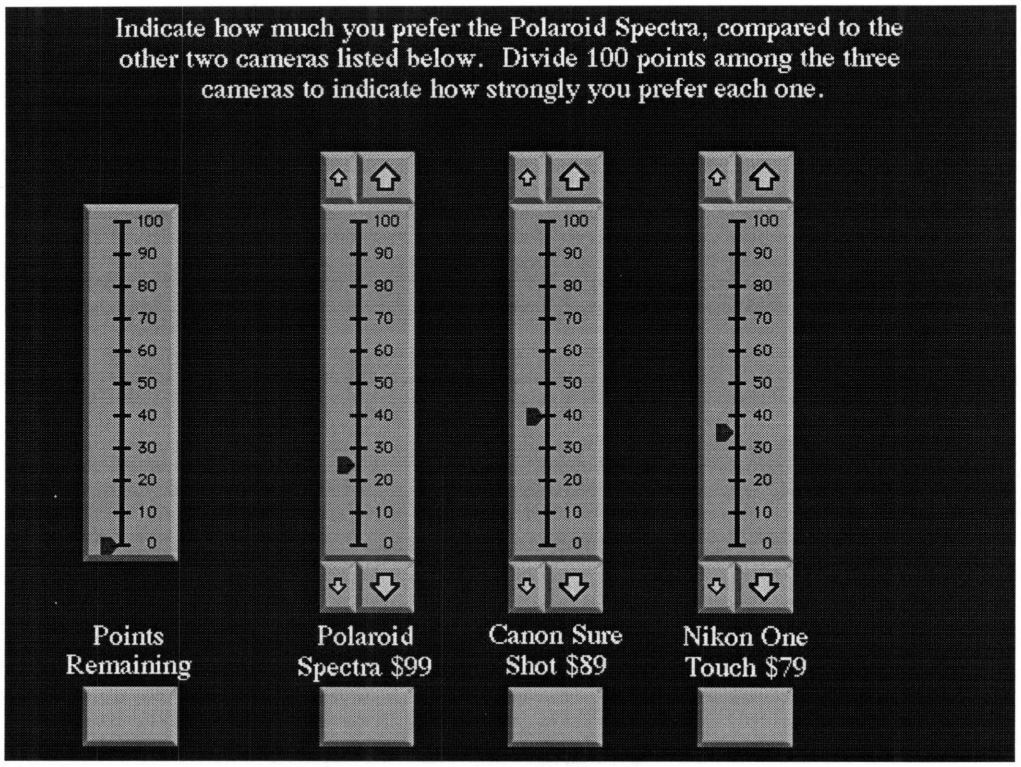

Indicate how much you prefer the Polaroid Spectra, compared to the other two cameras listed below. Divide 100 points among the three cameras to indicate how strongly you prefer each one.

Figure 4.1: Chip allocation example screen

To reduce clutter and complexity, each of the different questions usually gets its own "screen," unless several questions in a row are of the same type and can be combined. The screens use large icons, pleasing background colors, and intuitive visual clues to reduce the risk of confusion (Hsu, 1996). The ideal retail Web site, especially one attempting to use IA methodology to gain marketing information, should include some of these ideas in its interface.

The Task at Hand

The task to create a unique, interactive Web shopping experience dictated that I search for an implementation beyond plain-vanilla HTML pages. Many corporate Web sites used them and seemed to be little more than electronic catalogs, which consumers consulted only to get price and purchase information. We wanted, however, to attract users to our site and to keep them as long as possible by making it fun and interactive. I needed to design a flexible framework that would help us attain that goal.

Relevant Experience with IA Surveys

My previous multimedia software development experience aided me greatly in that task. After completing my senior thesis at the Media Lab, I worked on two IA projects during 1996, the EDF and the Salsa projects. When I took on the EDF project, it had mostly been finished, but I changed a few final specifications at the request of EDF before finally sending it to France. The Salsa survey, designed to study the effects of marketing on consumers' perceptions, shipped near the end of 1996. I wrote all of the code for it, and designed the screens and media to suit the survey questions.

Through programming both surveys, I learned much about the Apple

Media Language (AML). It was a very object-oriented language, even though its syntax was much like that of normal speech. It provided a framework of objects to handle multimedia easily and consistently. Display-sized containers called screens held and manipulated media elements such as images, text blocks, text fields, and movies. I could subclass any of the media elements to create non-standard effects.

Problems with AML and the Surveys

AML gave me power and flexibility, although it had its limitations. The first was that the programming environment seemed to need an upgrade with each new Macintosh operating system. Old surveys would not run on newer systems, while new surveys would not work with older systems. Furthermore, AML could not handle very complex tasks. It did provide hooks to external functions written in C, which meant that I could write C code for complicated routines, but they were unwieldy and often unpredictable. Lastly, the AML surveys would only work on the Mac or the PC, and creating one for both systems meant using the short, standardized ("8 dot 3") DOS filenames.

While these surveys were relatively easy to produce for use on single Macintoshes or PCs, they were limited by the chosen platform and the localized storage of data. The IA project for EDF, in particular, experienced problems with the latter limitation, partly due to assistants who conducted the surveys while not adequately understanding the data backup procedures. Web-based implementations, however, could prevent such mishaps in the future, since data files could be stored securely on the site's Web server itself, under the supervision of the original designers and programmers. In addition, with the aid of the Java programming language by Sun Microsystems, a programmer can now write a single application that can run on any computer with a Java-equipped Web browser installed.

Some Pros and Cons of Using the Web

Despite the benefits, however, using the Web is not an altogether easy solution to program distribution and data collection problems. While Java has revolutionized Web programming by providing powerful abstractions for passing information over the Internet, download time for complex Java applets (or "tiny" applications meant to be run within a Web browser) can be quite lengthy, perhaps up to ten minutes over poorer connections. Once the applet downloads, it runs reasonably quickly, but a typical Web surfer will probably not bother to wait ten minutes, even if the applet could be very useful. He may wish to get the same services from a different source without the wait, even if the alternate source is not as good. If a site must extensively use the power and flexibility of Java to accomplish its goals, rather than using plain HTML or forms, then it should also consider the potential problem of long download times and possible solutions.

Deciding How to Implement the Framework

When considering tools for building the framework, I originally avoided Java, because I had thought that it would be too slow, complicated, buggy, and ambitious for the prototype site. I thus began by studying plain HTML but found that it could not handle the data storage and calculations that we would need to provide an online shopping advisor. I also considered CGI scripts, since they could provide storage and perform calculations, but found that I could not easily implement interactive controls with them. I briefly looked at ActiveX, but since it was limited to Windows™-based machines and we wanted to reach as many users as possible, I discarded it from the consideration set. With no other viable options left at the time, I then researched the capabilities of Java.

Java fulfilled the most important objectives. It would allow me to create a unique interface with interactive controls, to store data, and to calculate best fits between customers' stated needs and available products. It did, however, complicate the design of the site. Furthermore, once I decided to create the entire interface in Java, the site became much less flexible to change than it would have been with a hybrid of Java and HTML. Java added power while it increased complexity, so building a site with it demanded careful use of modularity and abstraction. In addition, due to the aforementioned problems with long applet downloading times, I had to be even more wary.

An Initial Java Feasibility Test

Before I ventured too far with Java, I tried an initial feasibility test. I took the complicated chip allocation example "screen" from a multimedia survey (the same as in Figure 4.1) and attempted to port it to Java. The code to handle the mouse clicks in each scale, to calculate the values of each scale, and to move the value indicators was the most complex and lengthy of any code in the surveys, so it should have represented the most complicated functionality that the project would ever need.

The Java applet version performed beautifully. It loaded extremely quickly in both Netscape Navigator™ and Microsoft Internet Explorer™

and reacted immediately to every click, doing as well as the original version in AML on the same test machine. In addition, since Java simplified offscreen image buffering, I used it to do all of the drawing offscreen before displaying it, so the graphics updated without flashing. Thus, except for the browser window, the Java applet looked and acted just like the screen from the original survey in AML. I was very encouraged by the results and began to brainstorm a Java-based framework for the site.

Further Java Research

Before I could thoroughly design the framework, however, I needed to learn more about Java and its capabilities. I bought several books (referenced at the end of the thesis) and a Java compiler from Metrowerks, whose CodeWarrior™ compilers have repeatedly earned my trust. Sun's JavaSoft corporate Web site provided additional information and tips about Java's built-in libraries and linked to several other third-party sites with additional libraries for multimedia and database applications. With these references and resources at my disposal, I began to fully explore Java and understand what it could and could not do for an interactive Web site.

The evolution of the prototype design and its Java framework after the initial test can be found in Chapter Six. Chapter Seven summarizes the purposes of each of the classes, and the Appendix includes the full source code.

Quick Introduction to Java Basics

Syntax, Garbage Collection, and Basic Object Features

Java's syntax looks a lot like that of C++. A Java class has a constructor method (or function), like a C++ class, but has no true destructor. Since Java garbage collects its objects, unlike C++, it has no need of destructors that purge and delete extraneous objects. (If a programmer does want to aid garbage collection, however, he can override an object's *finalize()* method, which the garbage collector may or may not execute.) As with C++, the programmer can subclass any class and override any of its non-static and non-final methods.

Variable Types

For basic variable types, Java provides the *boolean* type and the Unicode 16-bit *char*acter type in addition to the usual sets of byte, integer, and floating-point types. To avoid ambiguity, the boolean type may only hold true or false values, instead of zero and non-zero values, as in C++. As an extra feature, newly declared variables are by default set equal to zero, false, or null, depending on the type. This feature allows the programmer to skip variable initialization in many instances that he might have otherwise forgotten and regretted if using C++.

Built-in Code Libraries

Java has many built-in classes, arranged in *packages.* The ones which I used for the basis of the framework are the following: *java.lang, java.awt, java.awt.image, and java.applet.* The *java.lang* package groups all of the language's basic classes, such as Object, Class, String, and Thread. The *java.awt* package (where *awt* stands for Abstract Windowing Toolkit) contains platform-independent interface classes, such as Event, Frame, Image, Button, Font, Menu, and Window. *Java.awt.image* provides image filter and color model classes, and *java.applet* holds the basic applet class.

More About the *java.awt* Package

Sun designed the *java.awt* package to be extremely platform-independent, so that even though windows, buttons, and menus may look different on PCs than on Macs, they can all be created with the same Java code and work correctly. Java accomplishes this feat through the use of *layout managers* which automatically size and place all common interface objects (*components*), such as buttons and checkboxes. The layout managers guarantee that all components will appear on the screen and will not overlap, despite the GUI differences from system to system, so that the programmer can rest assured that his application will look decent and will work correctly on any system that supports Java. Although different layout managers abstract placement to varying degrees, the programmer no longer needs to place components at precise pixel coordinates. (Some may argue that Sun overdesigned in this particular case, since programmers often want to specify precise coordinates to ensure a particular "look." In external windows other than a browser window, designating coordinates for components and having them take effect requires a few convoluted hacks.)

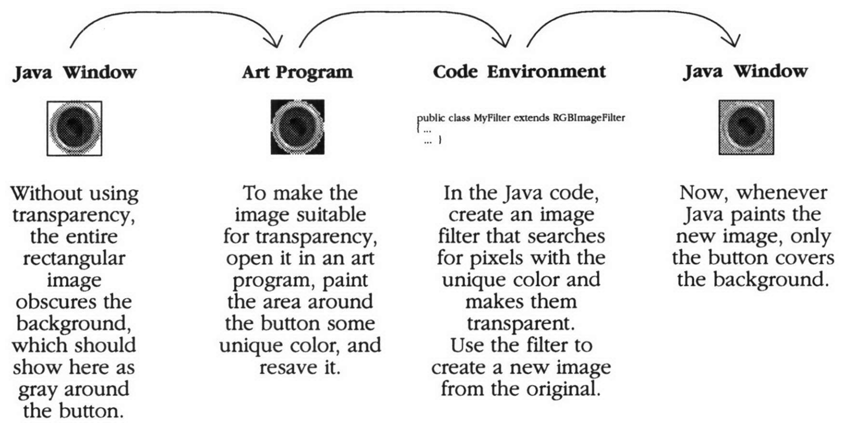
Containers in the AWT, such as panels, frames, and dialogs, are components that hold and organize other components. Since containers are also components, they can hold each another as well. The panel, a basic container, only helps to organize other components within a window or another panel. The frame, however, is a fully functioning window, complete with a title bar and pull-down menus. The dialog fills in the gap between the frame and the panel. It is a small pop-up window often used for confirmation requests or temporary messages.

The complete list of basic AWT components are the following: buttons, labels, checkboxes, radio buttons, lists, choices, text fields, text areas, menus, canvases, and scrollbars. While most of their names describe them adequately, a few are less clear. Labels are simple text strings that serve no other purpose than displaying themselves on the screen. Lists provide scrolling lists of items, similar to a file selection list, and allow the user to select single or multiple items from the set. Choices are simply pop-up menus of textual items. Lastly, canvases have no special functionality but give the programmer flexibility to create custom graphic components by overriding their basic event handling and painting methods.

The *java.awt.image* Package and Image Effects

The filters in the *java.awt.image* package can crop images and create interesting image effects, including color cycling and transparency. For example, I can create with an art application a non-rectangular object that should be painted seamlessly over some background. (Typically, Java would paint a rectangular image to the screen, and any non-rectangular object in the image would have a visible border.) If the art application does not support transparency, I can surround the object with some color that it does not use, such as pure green, and save the image. Using a color filter from *java.awt.image*, I can then filter the image for shades near pure green and convert those pixels from opaque to completely transparent. Java will then recognize the transparent pixels whenever it draws the new image, and the object should appear to lay seamlessly over the background.

As an example, the figure below shows each of the steps. The natural background image in the figure is a 50% gray tone. The color used to signify transparency for the filter (such as pure green) appears here as 100% black.



```
public class MyFilter extends RGBImageFilter
{ ...
  ... }
```

Without using transparency, the entire rectangular image obscures the background, which should show here as gray around the button.

To make the image suitable for transparency, open it in an art program, paint the area around the button some unique color, and resave it.

In the Java code, create an image filter that searches for pixels with the unique color and makes them transparent. Use the filter to create a new image from the original.

Now, whenever Java paints the new image, only the button covers the background.

Figure 5.1: A method of creating transparent regions in Java images

The Evolution of the Framework

Early Conflicts with the AWT Layout Managers

Early during the initial planning, I concluded that I would need to circumvent the AWT layout managers to achieve the "look" that we wanted for the site. I found as I created some example question screens with buttons and text fields that consistency was hard to achieve with the layout managers. I could not place the components anywhere near where I had intended over the background image, despite my choice of layout manager.

Since we wanted to layout the components ourselves in an exact way so that we could guarantee they would look the same on any system, the layout managers presented a problem. Besides, in my earlier experience with multimedia surveys, I had custom-built and placed components on the survey screens so that they would look as good as possible. I had hoped to be able to build on my successful survey programming experiences while designing this site. I wanted to find a way to avoid using the built-in layout managers, despite their advantages.

While working with test applets inside the browser window, I was able to disable the layout managers by setting each applet's layout manager equal to null. I could then specify exact coordinates for the components and have them take effect. Since this worked so well for all of the components in the browser window, I began to design a framework for putting the interface in an external applet window (or frame). (I would discover later, however, that the same trick did not work within applet windows that were separate from the browser.)

The Original Object Hierarchy and Its Problems

The original object hierarchy (in terms of containment and functionality) looked much like that shown below in Figure 6.1. The parent applet that would run inside the browser window created the independent frame and the child applet that would run inside the frame. Screens could be added to the frame as panels, and hold other components, such as buttons, choices, text fields, and the dashboard canvas object (which existed because we originally wanted the interface to look like a car dashboard

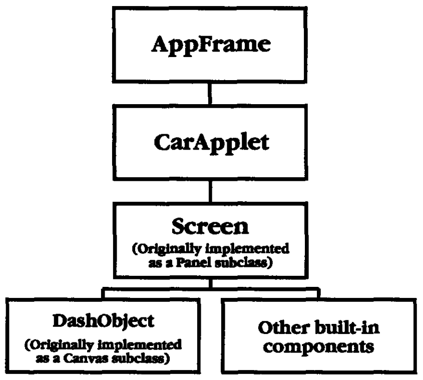
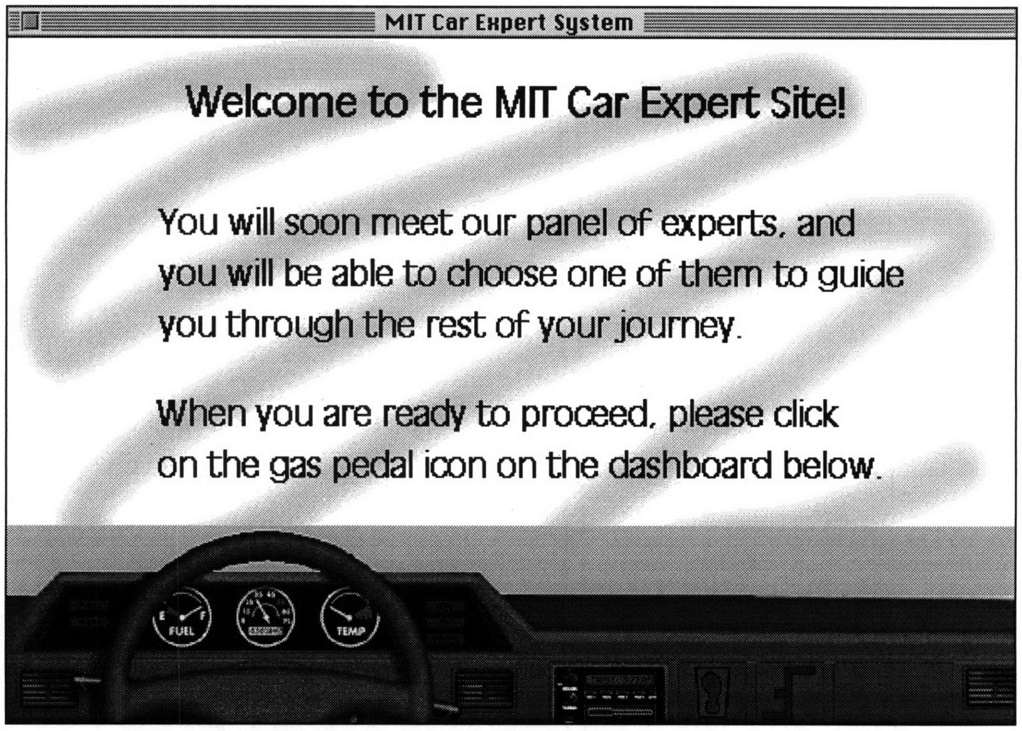

AppFrame
CarApplet
Screen
(Originally implemented as a Panel subclass)
DashObject
(Originally implemented as a Canvas subclass)
Other built-in components

Figure 6.1: Original object hierarchy, by containment and functionality

and windshield). We adopted the *prototype* dashboard image from the AutoByTel Web site. The original test looked like Figure 6.2 below.



Welcome to the MIT Car Expert Site!

You will soon meet our panel of experts, and you will be able to choose one of them to guide you through the rest of your journey.

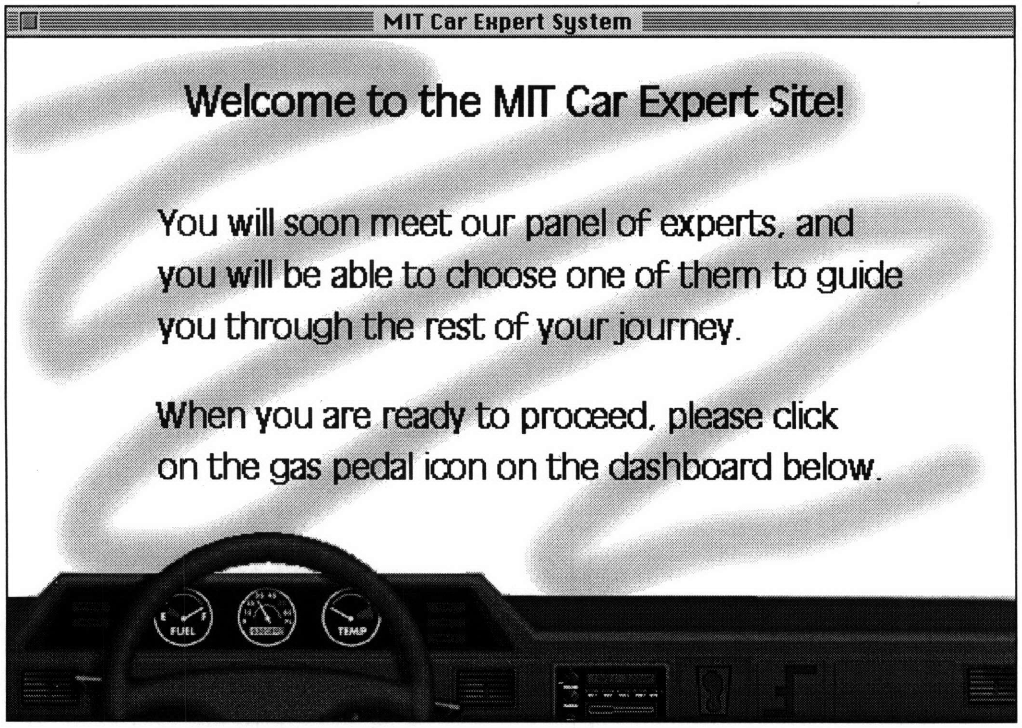
When you are ready to proceed, please click on the gas pedal icon on the dashboard below.

Figure 6.2: First test with the dashboard object as a Canvas subclass

While the hierarchy worked, I had overlooked that canvases could not overlap transparently over panels (using the then current Java AWT libraries). While I had used an image filter to make the border transparent around the non-rectangular dashboard image, I could only see the background of the canvas behind the dashboard. As displayed above in Figure 6.2, the canvas unfortunately covered the image on the screen panel underneath it.

A couple of solutions to the transparency problem presented themselves, although neither were attractive. I could detract slightly from the modularity and use the dashboard image without containing it in a canvas, or I could paint the background screen image within the canvas before painting the dashboard image in it. Since at that time I had never tried to crop images in Java, and since I expected that the extra step would add a noticeable delay to screen updates, I chose the former solution.

The dashboard canvas disappeared and became an Object subclass. The new dashboard could contain an array of images and could either be enabled or disabled by the parent screen. Depending on the dashboard's



Welcome to the MIT Car Expert Site!

You will soon meet our panel of experts, and you will be able to choose one of them to guide you through the rest of your journey.

When you are ready to proceed, please click on the gas pedal icon on the dashboard below.

Figure 6.3: The dashboard as a simple Object with a transparent image

status, its images could be drawn or ignored during drawing updates. The solution worked well, as shown in Figure 6.3. Until we discarded the dashboard interface several months later, it remained as described.

Screen Sequences and Memory Problems

Continuing from that success, I began adding example screens containing just background images so that I could test the correctness of the screen transition code. After a few minor bug fixes, the screens seemed to flow well from one to another. Each screen preloaded the following screen, so that it would display quickly whenever the user decided to continue, and each screen removed all records of itself whenever it disappeared, so that it would be garbage collected. The first few screens transitioned quickly, but later screens began to change over much more slowly, until the application finally died. I added some code to display available memory at the end of each transition and found a memory leak with each new screen. The application always ran out of available memory and crashed, despite the choice of the test platform or the browser. What had initially looked like success had instead become another roadblock.

I scoured the code for the removal of each screen to make sure that all records of the screen and its images would be garbage collected. The code was very consistent, and should have cleared all traces of the passing screens. The problem mystified me for about two or three weeks as I looked for solutions. The answer finally came from Sun's documentation for the Image class.

The AWT documentation revealed that all images have a *flushO* method that flushes all resources that they might hold. The full description was as follows:

> "public abstract void flush()
>
> Flushes all resources being used by this Image object.
>
> These resources include any pixel data that is being cached for rendering to the screen as well as any system resources that are being used to store data or pixels for the image.
>
> The Image object is reset to a state similar to when it was first created so that if it is again rendered, the image data must be recreated or fetched again from its source."

The facts that images cache their data with system resources and that the *flushO* method exists suggested that garbage collection of the Image object might not touch the cached data. Indeed, some tests with a few rotating screens showed that the first pass through the set was slow due to image loading but additional passes that should have been just as slow proceeded much more quickly. I added the *flush()* call to the code for the removal of each screen, and retested the long sequence of screens.

The test sequence completed without any errors. The *flush()* method solved the memory leak problem. I then continued to optimize the various modules and began to test interface components inside the external applet window.

Further Conflicts with the AWT Layout Managers

The first component that I tried came from a third-party library for partially interactive three-dimensional image buttons. I had read about them and decided to try one for the gas pedal/continue button in the dashboard. The library's programmer had chosen to implement the buttons as canvases. The canvases could intercept mouse events and could paint one of a set of button images with an arbitrarily thick three-dimensional border, depending on the button state. The original trial looked good (Figure 6.4), but I had great difficulty placing the component in the window with exact coordinates.

At first, with its container's layout manager disabled, the button would not appear on the screen, even though it had definitely been added to the

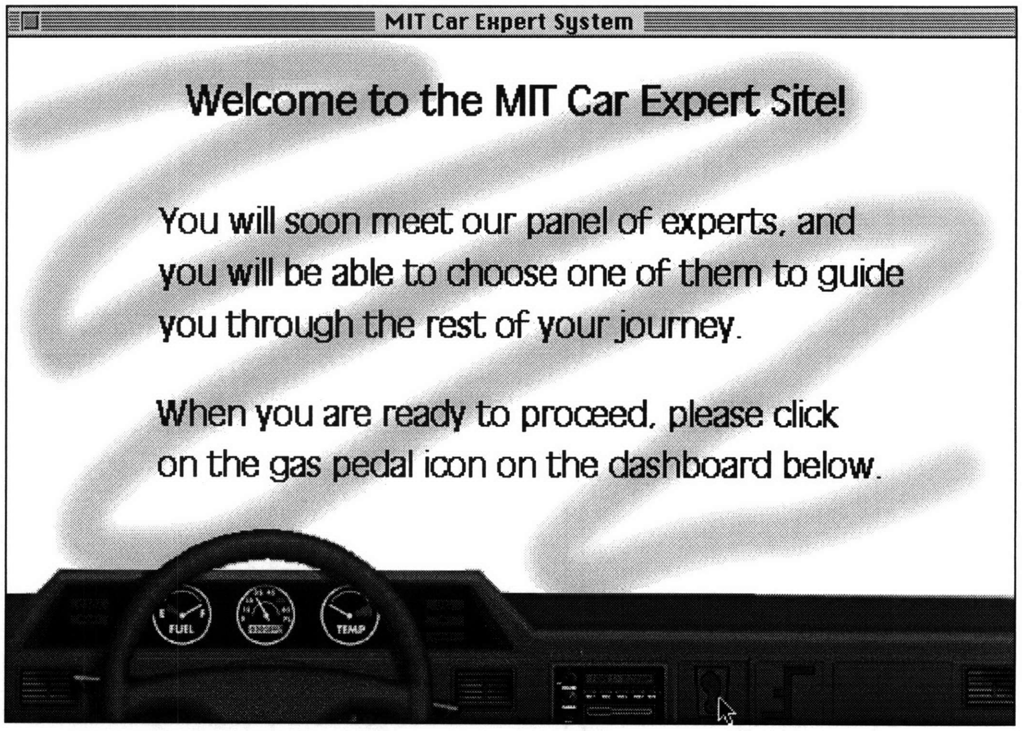

MIT Car Expert System
Welcome to the MIT Car Expert Site!
You will soon meet our panel of experts, and you will be able to choose one of them to guide you through the rest of your journey.
When you are ready to proceed, please click on the gas pedal icon on the dashboard below.
FUEL
TEMP

Figure 6.4: Dashboard with gas pedal as a three-dimensional image button

container. When I reenabled the container's layout manager, the button appeared in the correct place, but would not remain there. Rather inelegantly, I responded to that by overriding the button's painting and layout methods to force it to reset its coordinates with every update. It worked, but I as tested some basic components a week later, I found that they too suffered from the same problem. I thus had to find a better and more elegant solution than subclassing and overriding painting and layout methods for every basic component.

Since the components required a layout manager inside the application window, the easiest compromise was to create a new layout manager that would accept custom-placed components. As an initial test, I subclassed the simplest layout manager, FlowLayout, named it CarLayout, and overrode its *layoutContainer()* method. A container using CarLayout would call the new *layoutContainer()* method whenever it needed to layout all of its components. The first version of the new method looped through all of the container's components and called *ReshapeButton()* for any three-dimensional image buttons. With CarLayout as the layout manager for its container, the gas pedal appeared where it belonged in the window and remained there from screen to screen. (The code for the original CarLayout layout manager is in the Appendix and is currently being extended to apply to other basic components.)

Scrapping the Dashboard for a Better Interface

During this period of testing and optimization the marketing design team and I had not yet cemented the final look of the user interface. We disliked the gas pedal button, since its placement in the dashboard was utterly non-intuitive, but we could not agree on a better place for it. Although we wanted to continue the dashboard/windshield metaphor, the gas pedal had to go. We finally met to brainstorm a new interface design.

At the meeting, we introduced both an acceptable modification to the original design and a completely novel idea, based on role playing games (RPGs). Since we concluded that proceeding with the RPG idea without first doing marketing studies was a bit risky, we decided to modify the old design for our June prototype and to perform the necessary marketing studies over the summer.

After a few days of work, the modified interface eventually appeared as shown below in Figure 6.5. With the help of Photoshop™ filters, the "artistic" windshield became a cloudy, blue sky. The dashboard disappeared completely, and a dual-directional GO button replaced the gas

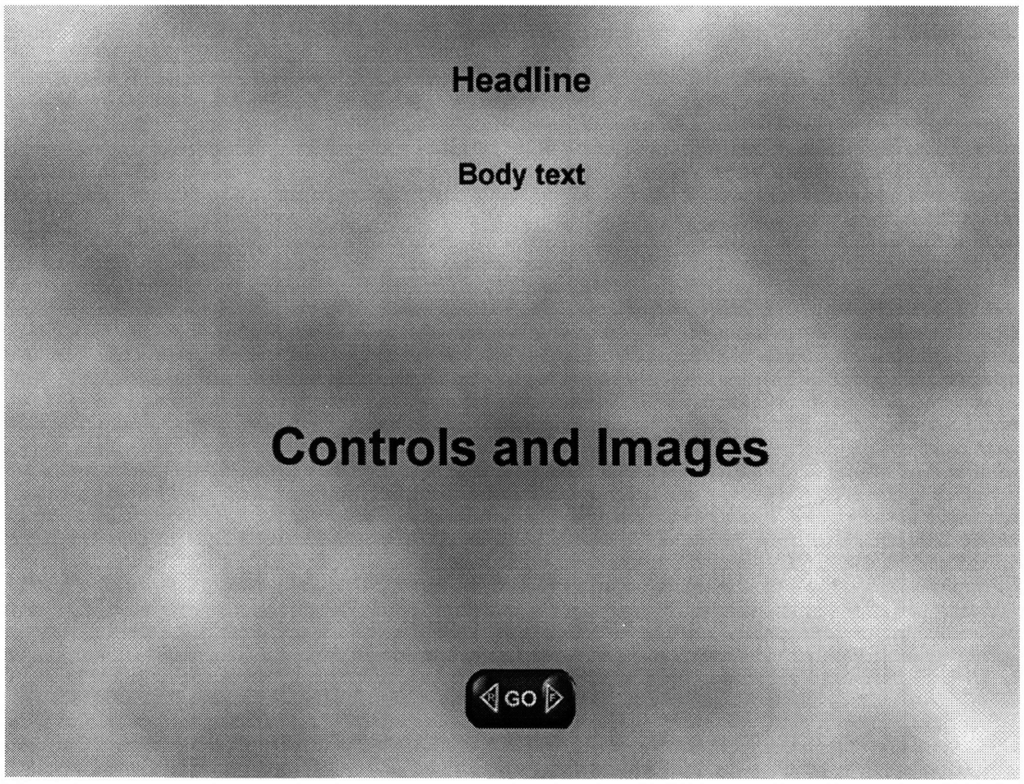


Figure 6.5: Template for each screen's interface

pedal as the screen transition button. Since we wanted the GO button to look as realistic as possible to match the new background, I digitized a SEEK button from a picture of a General Motors dashboard and altered it in Photoshop™ to look like the GO button as shown below.

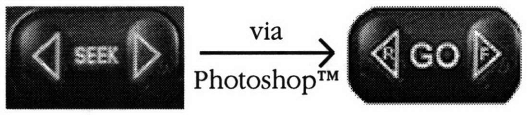


Figure 6.6: Transformation of the SEEK dashboard button to GO

Design of Better, More Flexible Image Buttons

After designing the GO button, however, I realized that the three-dimensional image button class that I had been using would not be sufficient for it. Since the GO button was non-rectangular and since it needed to be able to handle requests for both reverse and forward directions, the image button class could not satisfy it. Even if it had been subclassed and extended, the old image button class could not handle the transparency and additional button states without an extensive overhaul. I thus had to devise a new image button class.

Since I already had an ImageButton class, I named the new class MyButton. I designed it to be even more flexible than the third-party three-dimensional image buttons, so that transparency would work correctly, and so that any button could handle an arbitrary number of states. Each MyButton included its background image and an array of images for its states. It also had its own image filters and variables for its current state, origin, and dimensions.

By this time, I had become extremely familiar with Java's syntax, strength, and power. The MyButton class worked well for the GoButton class, which only needed to add a few extra methods to handle the extra status of the two different directions and to notify the frame that the user had requested a screen transition. As with buttons to follow, it also overrode the following three standard methods: *InsertButtonImages()*, *PaintButton()*, and *ChangedState()*. The MyButton class provided enough abstractions that only those three methods needed to be overridden for most simple buttons.

Next, I wanted to create custom, interactive radio and checkbox buttons for the site using the same ideas behind GoButton. I subclassed MyButton to create the RadioButton and CheckBoxButton classes. While the *ChangedState()* methods for both classes were a bit complex, debugging them took little time. As with GoButton, they only needed to override *InsertButtonImages()*, *PaintButton()*, and *ChangedState()*. Except for the original implementations being a bit too large, they functioned as desired and looked great.

Work in Progress

I began to write this section for the thesis after finishing the custom button classes, and left in progress the following classes: Databank and ChipModule. Databank would collect and store the data from the components of each screen for use with the expert advisor algorithm, and ChipModule would provide a flexible abstraction for adding the 100-point "chip allocation" module to a screen. At the same time, I also researched third-party database frameworks to hold all of the car specifications for the prototype's expert advisor algorithm. All of these would be finished before the demonstration of the first prototype in June 1997, since the prototype would need the database and the Databank module to show an actual best-fit calculation.

The next chapter summarizes each of the classes that I wrote or borrowed from third-party sources, and the Appendix includes the entire code library as it existed while I finished this thesis.

Java Framework Class Descriptions

Class Hierarchy

The framework at the end of May 1997 contained all of the classes shown in Figure 7.1 below. The hierarchy of the diagram represents the geneology of the classes, with subclasses drawn below their ancestors. All of the classes that had been scrapped by the end of May are not included. The italicized class names in the diagram are built-in Java classes, the bold names are my classes, and the plain-text names are third-party classes.

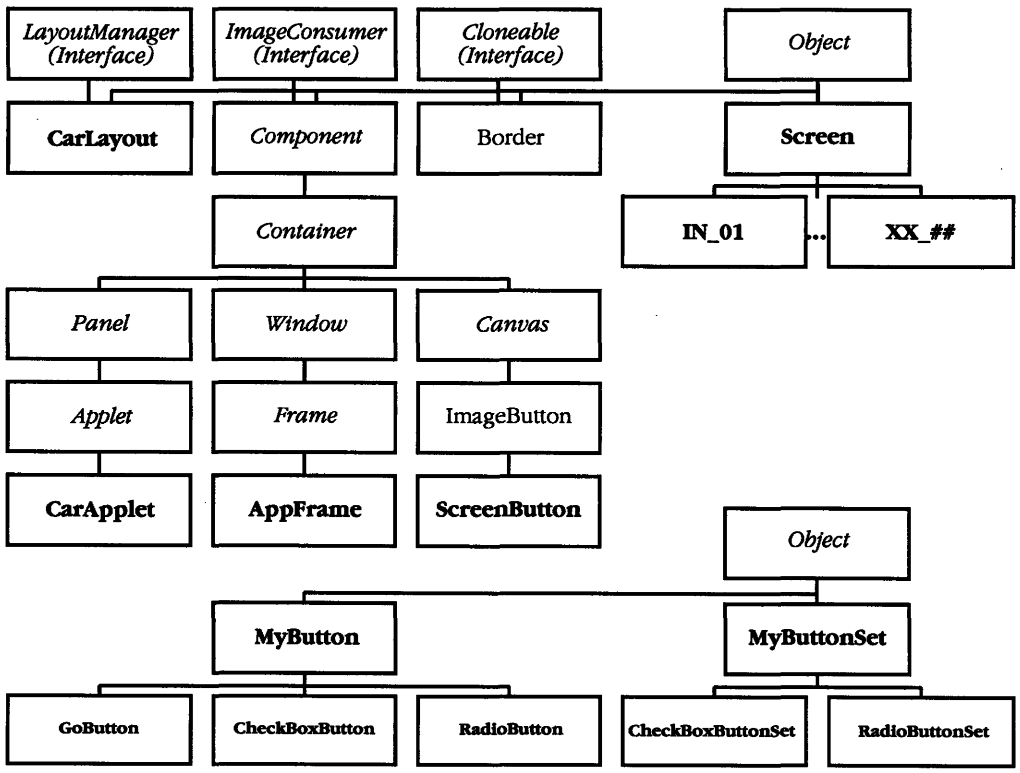


Figure 7.1: Class inheritance hierarchy
(Italic: Java base classes; bold: my extensions; plain-text: third-party extensions)

The next figure shows the main subset of the classes grouped by function and hierarchical placement. Objects contained by other objects appear below their containers. For example, components placed within screens are grouped below screens. Screens then fall under the applet frame, and so on.

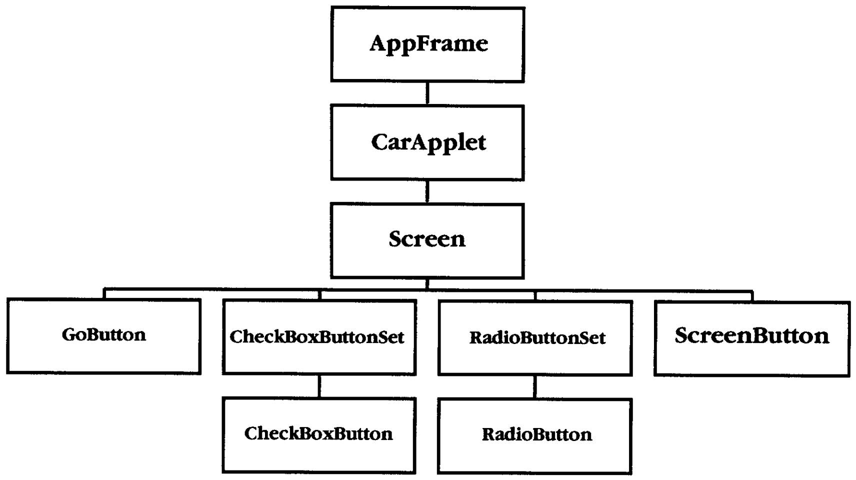


Figure 7.2: Class containment hierarchy

Class Functionality Descriptions

Descriptions of the classes follow. Complete source code for the classes is given in the Appendix.

CarApplet

The CarApplet class extended the Applet class. Its main functions were to initialize the main applet frame and to place a copy of itself inside the frame. The copy (or *child applet*) ran inside the frame and took responsibility for coordinating the redrawing of everything within the frame.

CarLayout

The CarLayout class extended the FlowLayout layout manager class. Its job was to ensure that components consistently placed themselves at exact pixel coordinates within their containers. It provided only one important method, *layoutContainer()*.

DashImageFilter

I originally designed the DashImageFilter class for the dashboard, but even though the dashboard concept died, the filter remained useful. Like the CarLayout class, the DashImageFilter has only one responsibility and one major method. DashImageFilter extended the RGBImageFilter class and overrode its *filterRGB()* method to change opaque, nearly pure green pixels in an image from opaque to transparent. Practically every custom-made, non-rectangular interface item used this class.

AppFrame

The AppFrame class extended the basic Frame class. Since I would only need one instance of the frame at a time, I piled a lot of functionality into it to save valuable memory space. It was thus one of the largest classes and provided many useful methods. The AppFrame held the child applet and any components added to it by the screens. It provided essential methods that performed screen transitions, distributed mouse events, loaded images, and added items to special lists.

Screen (and its subclasses)

The Screen class simply subclassed Object. It contained variables to reference the last and next screens, to indicate if it had already been visited, to hold an array of images, and to affect the behavior of the GO button. Its methods provided the means to change any of the variables, to add or remove buttons from the applet within the frame, to add concurrent screens to the frame for preloading, to paint all images, and to flush all images (including those of buttons) during screen transitions. It also included a method to compare a new screen subclass with the next screen, so that if the next screen had already been visited and the two subclasses were equivalent types the applet would not initialize another copy of the screen.

Subclasses of the Screen class, such as IN_01 (the first introduction screen) and EX_01 (the first expert advisor screen), usually overrode only the *PresetCharacteristics()* and the *AddAllNextScreens()* methods. The former would initialize all of the images that the screen would need, and the latter, in addition to providing the frame with an initialized list of all possible following screens, would finally add the images and any needed components for display. Athough I had not included the code by the end of May 1997, I had also planned a standard method to override for use during screen transitions to store component state data. (The data would first determine the choice of the next screen, if there were several possible choices, and it would later support the expert advisor's vehicle selection algorithm.)

Border and ImageButton (third-party)

The Border and the ImageButton classes came from DTAI, Incorporated. The ImageButton class extended the Canvas class to provide three-dimensional buttons with images instead of text. Border worked with ImageButton, and handled the allocation and drawing of an arbitrarily thick three-dimensional border around an ImageButton's current image.

An ImageButton could choose among different images for each of its states, so that disabled buttons could look different from enabled buttons, and so on.

ScreenButton

The ScreenButton class extended the ImageButton class to make it suitable for use with screens. It initially included code to update the button in the frame's offscreen image buffer, rather than onscreen. Since the button did not have "on" or "off" states like radio buttons or checkboxes, it later provided a method to notify its "parent" screen of clicks.

MyButton

The MyButton class supplemented the ImageButton class. It provided a flexible framework for highly interactive image buttons. It only subclassed Object, not Canvas, so it fully supported image transparency over screen backgrounds and thus permitted non-rectangular buttons. MyButton contained methods to initialize its background, images, companion buttons, filters, and state. It also had methods to determine if a mouse event occurred inside of it and if the event changed its state, to paint an image corresponding to its current state, and to flush its images when removed by its parent screen. MyButton optimized updates due to state changes by providing *UpdateButton()*, which would only redraw the rectangular region of the background image behind the button instead of the entire background.

MyButtonSet

The MyButtonSet class allowed grouping of similar MyButton instances. It extended Object to hold arrays for a set of MyButtons and their origins. It first initialized a single button with all of its images and then “cloned” it as many times as desired to complete a set. The clones would all reference the same images as the first button, so only one set of the images would eat up memory, instead of *n* sets, where *n* would be the number of buttons. MyButtonSet would then loop through the array of initialized buttons and make them all companions, since they all belonged to the same set. In addition, since MyButtonSet grouped individual buttons in a higher-level set, it provided an abstraction method to inquire if any of its buttons had changed state, so the applet frame could distribute the mouse events to whole sets instead of individual buttons. It also gave an abstraction method for painting all of its buttons, so that the applet could redraw all the buttons in a set with one call.

CheckBoxButton

The CheckBoxButton class extended MyButton. It defined the eight following states that a fully interactive checkbox button would require: off, on, off with mouse over, on with mouse over, off and pushed, on and pushed, off and disabled, and on and disabled. To create the images for the individual states of this button (as well as the other buttons described below), I took a button from a General Motors dashboard picture and altered it with Photoshop™. CheckBoxButton overrode the *InsertButtonImages()* method of MyButton to load its specific images, the *PaintButton()* method to paint the correct image with each state, and the *ChangedState()* method to specify its particular state transition behavior.

CheckBoxButtonSet

The CheckBoxButtonSet class extended MyButtonSet to provide a container for checkbox buttons. It overrode *InitOriginalButton()* and *InitClones()* to suit the specific initialization requirements of checkbox buttons.

RadioButton

The RadioButton class subclassed MyButton much like the CheckBoxButton class. It included the same eight states, but needed additional message handling capabilities, since radio buttons in a set are mutually exclusive. Thus, in addition to the methods that CheckBoxButton also overrode, RadioButton redefined *HandleMessage()* and used *NotifyCompanions()* within *ChangedState()* to allow radio buttons within a set to communicate. For the images, I again took a button from a General Motors dashboard picture and altered it with Photoshop™.

RadioButtonSet

The RadioButtonSet class extended MyButtonSet to provide a container for radio buttons. Like the CheckBoxButtonSet, it overrode *InitOriginalButton()* and *InitClones()* to satisfy the particular initialization requirements of radio buttons.

GoButton

Lastly, the GoButton class subclassed MyButton. GoButton defined twelve specific states, needing four more than the other buttons since it was a bidirectional switch. To be fully interactive, it had to display mouse events over both the reverse or forward sides of the switch. Since some screens would not allow the user to move in both directions, it also provided status variables and methods to separately enable or disable the reverse and forward sides of the switch. As with the other button types, it overrode the usual image initialization and state description methods. Lastly and most importantly, it included the *NotifyGo()* method to inform the applet frame whenever a user clicked it so that the frame could initiate an appropriate screen transition.

Classes in Progress

Classes in progress at the end of May 1997 included the ChipModule class, an all-purpose, interactive, 100-point "chip allocation" module designed for marketing questions, and the Databank class, a convenient repository for all of the screens' component state data. I intended the ChipModule class to be extremely flexible, providing a single abstraction to permit easy management of a variable number of sliding scales. The scales themselves could be any uniform size and could be placed arbitrarily far apart.

In contrast to the complex ChipModule, the Databank class would be very simple, containing only large arrays of records and the methods needed to access them. Each screen that collected component data would call the main Databank instance with its data while transitioning. Methods in Databank would then use the particular screen subclass as an index into the array of records and would store the data in the appropriate record.

Reflections and Future Research Directions

Lessons Learned

Working on this project taught me four important lessons. The first was to never rush to implement a software framework before fully understanding the basics. I discovered this bit of wisdom two months into the development of the project, after I had spent a couple of weeks reorganizing and repairing my class framework.

The second lesson was to never assume anything about Java. Since few programmers at the time had written such complex Java applets, I could barely find texts that discussed more than the basics. I took what examples I could find and extrapolated, assuming that everything would work as it had on a smaller scale. The memory problems I had with images, however, proved otherwise. It took me so long to uncover the cause of the memory leaks that I learned to test each of my assumptions separately before combining them with the framework.

The third lesson was to require specifications to be as concrete as possible before implementing them. I spent countless hours rewriting and repairing my framework after each revision of the specifications. If I had waited until they stabilized before continuing or had forced the issue earlier with the rest of the team, I might not have had to change as much code as I did.

The fourth lesson was to build a flexible system. Having concrete specifications at the very beginning of a project is a engineer's dream; rarely is it a reality. Since most designs transform as they develop, a programmer should provide enough abstraction so that he will only need to change a few small modules instead of an entire framework. Only as I designed my last few classes did I begin to appreciate the efficiency of abstractions.

If I had known at the beginning what I know now, I might not have chosen to use Java for the prototype. When I began, I knew little about the competing technologies that might have provided easier alternatives. Repeatedly during the development of the framework I wondered if Java was worth the trouble. CGI scripts could have collected, stored, and calculated the data we needed, and HTML pages could have presented a limited interface. I often felt that I might have been a bit ambitious to choose Java, an emerging and unproven technology.

That feeling built as I had to move the project from our Apple Macintosh™ 8100/80s to a PC with a 200 MHz Pentium Pro™. At some point, the Java interpreter for the Macs could no longer handle the demands of the project. By the end of May 1997, only Microsoft Internet Explorer™ on the PC ran the applet as it was envisioned. Netscape Navigator™ could not redraw the partially transparent buttons as quickly or as unnoticeably as Explorer, which updated them almost instantaneously.

Since one of Java's hallmarks is its compatibility with all major systems, I felt some regret that the project only ran well on the best systems. The concepts behind it were great, but they were perhaps a bit ahead of their time. My teammates reminded me, however, that I was building a prototype for a site three years in advance. During that time, computers, network connections, and Java interpreters would likely improve so that most users would be able to enjoy the site as it was intended.

Further Java Development and Interface Ideas

Due to all of the roadblocks that I encountered, the prototype site left many stones unturned. With more time, I could have included more complicated Java classes and dealt with efficient client-server communication, data storage, and animation sequences. The team might have also considered adding more trust cues to the interface and tried less conventional approaches to online marketing.

During one particularly eventful meeting, Frank Days, Salman Khan (an undergraduate researcher), and I brainstormed interface ideas. One that we came up with but temporarily discarded was called "Trucktown." A user entering the site would feel as if he were driving into Trucktown. The interface would suggest a small rural town, complete with streets, buildings, and people. The user would meet a friend upon entering the town and would use a map, similar to those found in video games, to navigate to any building within the town.

All of Trucktown's buildings would have special purposes, from the Town Hall, where the user might find preliminary information, to the Café, where he could interactively chat with other online users. Trucktown's residents included automobile experts, salespeople, librarians, and "normal everyday folk." Anything that the user could possibly want while shopping for a truck he could find inside Trucktown.

The Possibilities of Role Playing Games

The Trucktown concept implied the role playing genre of video games (RPGs). In RPGs, the user becomes a "persona" in some make-believe context, often portrayed as another country, world, or planet far away in place or time. Younger consumers, perhaps in their mid-twenties, have grown up understanding and enjoying RPGs. The young consumers with Internet connections typically have a reasonable amount of money to spend but have little patience with boring retail Web sites, especially when they contain nothing but redundant pages of text. A site which could successfully wrap its marketing in a RPG could perhaps become very popular with consumers in that segment and keep them interested in the site long enough to consider purchasing a product.

If the RPG idea could work well for the younger generation, what about the other segments who do not like science-fiction? When we first thought about that, we responded with Trucktown, which we thought would appeal to an older generation. Further ideas suitable for other segments (and for products other than automobiles) include Earth cities and towns, magical fantasy worlds, swashbuckling adventure worlds, and many others. For example, children enjoy stories and they like cartoon characters. To suit them, a site could have a make-believe world with talking animals and a storyline with "good guys" and "bad guys." Any one of these ideas could be expanded to appeal to various consumer segments and keep them "glued" to the site.

Trust and Role Playing Games

The RPG concept, however, does not address the trust issue very well. We originally rejected the Trucktown idea because we wondered if we, as consumers, would actually trust a RPG site. For example, would a consumer trust a cartoon more than a real person? Would a synthetic image inspire more trust than a real picture? We did not know the answers to those and other similar questions. Future marketing studies with focus groups and surveys could test the RPG concept to determine if users would enjoy playing with the system but not trust it enough to consider making a purchase from it.

Current Limitations

The RPG concept requires sophisticated computational power. While a Trucktown prototype could be implemented with Java now, the obvious extensions to animated figures, artificial intelligence, and high-quality two-way communication require several more years of technological improvements. Further research for electrical engineers and computer scientists could include studying the evolving systems, languages, and tools that could someday provide the required improvements.

Appendix A: List of Trust Cue Ideas

(This list comes from A. Benabadji's notes for a meeting in early May 1997.)

Channel-Based Trust

Concept

Description and justification of process
Endorsement of past users
Endorsement of a celebrity
Awards
Clear pricing policy
"Clean interface"

Guarantees

Brand name
Satisfaction guaranteed
External audits

Dependability

Identity of the system's sponsors
Statement of independence from manufacturers

Control

Security of payments
Privacy assurance
Interactive storytelling
Non-linearity (menu, sub-menu, etc.)
Selection of expert
Selection of microcommunity
User-friendly interface
Ability to return later

Expert-Based Trust

Credibility

Professional appearance
Description of expert and area of expertise
Acknowledgment of weaknesses

Objectives, Motivations, and Incentives

Explanation of motives and incentives
Explanation of deliverable

Microcommunities

Avatars
Background of people
Ratings by users of messages posted by other users
User-driven chat room
Scheduled chat conferences

Information-Based Trust

Communication Style

No time pressure
Simple language
Ability to change an answer
No personal questions until later
No price discussion until later
User can quit easily
First name basis

Fairness/Openness

Tells the good and the bad
Acknowledges not knowing the answer to a question
Provides links to other sites

Understanding the Customer's Needs

Frequent restatement of needs
Open-ended questions
Questions are based on earlier answers
User leads the whole process of questioning
User defines how he wants the information displayed and what data he wants

Appendix B: Java Source Code

```
// CarApplet.java
// This class provides the applet in the browser
//  and within the frame.  It is responsible
//  for all redrawing within the frame.

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

public class CarApplet extends Applet {
    boolean             standalone = false;  // Supposed to indicate if we're
                                             // the first or second applet.
    boolean             gotFrame = false;    // True when we have a frame.
    AppFrame            theFrame = null;     // References the current frame.
    int                 desiredW = 720; // Size is defined here (and only here).
    int                 desiredH = 540; //  Other references come from these.
    CarApplet           parentApp = null;    // References the first applet.
    Databank            storage = null;      // References the global databank.

    // Initializes the applet
    public void init()
    {
        super.init();
        if(!standalone)
        {
            // The parent is the first applet to run.  It
            //  has to allocate and open the frame, make a
            //  copy of itself, and stick it inside the frame.
            parentApp = this;
            init2();
        }
    }

    // On refreshes, if we're the parent and we have no frame,
    // make one.
    public void start()
    {
        super.start();
        if((parentApp == this) && (gotFrame == false))
            init2();
    }


    public boolean GetStandalone()
    {
        return standalone;
    }

    public void SetStandalone(boolean sa)
    {
        standalone = sa;
    }


    public AppFrame GetFrame()
    {
        return theFrame;
    }

    public boolean GetHasFrame()
```

```
    {
        return gotFrame;
    }

    public void HasFrame(boolean hf)
    {
        gotFrame = hf;
    }

    public void SetFrame(AppFrame inF)
    {
        theFrame = inF;
        HasFrame(inF != null);
    }


    public int GetDesiredWidth()
    {
        return desiredW;
    }

    public int GetDesiredHeight()
    {
        return desiredH;
    }


    public void SetParentApp(CarApplet theP)
    {
        parentApp = theP;
    }


    // only parentApp calls init2()
    private void init2()
    {
        SetStandalone(true);
        if(theFrame != null)
            return;

        storage = new Databank();

        CarApplet       cApplet = new CarApplet();
        if(cApplet != null) {
            theFrame = new AppFrame(this, cApplet, "MIT Car Expert System",
                                    desiredW, desiredH);
            if(theFrame != null) {
                gotFrame = true;

                theFrame.show();
            }
        }
    }

    // Paint Screen images first and then added components.
    public void paint(Graphics g)
    {
        if((this != parentApp) && (theFrame != null))
        {
            // Get and paint all images for the current screen.
            Screen  currentScreen = theFrame.GetCurrentScreen();
            if(currentScreen != null)
```

```
                currentScreen.PaintImages(g);

            // If we have a go button, paint it.
            GoButton    goBtn = theFrame.GetGoButton();
            if(goBtn != null)
                goBtn.PaintButton(g);

            // If we have any "MyButton" sets, paint them too.
            if(currentScreen != null)
                currentScreen.PaintAllMyButtonSets(g);

            // Be courteous and call the parent's method.
            // (Probably not necessary)
            super.paint(g);
        }
    }

    // Called whenever the applet needs to update itself.
    // If it's the child (second) applet, call the above
    //  paint method with the offscreen buffer as the
    //  Graphics argument.
    public void update(Graphics g)
    {
        if((this != parentApp) && (theFrame != null))
        {
            if(theFrame.offImage != null)
            {
                Graphics offG = theFrame.offImage.getGraphics();

                paint(offG);    // paint Screen images, Dash images, and components
                g.drawImage(theFrame.offImage, 0, 0, this);
            }
        }
    }

    public Databank GetDatabank()
    {
        return storage;
    }

    // This is called when packing/laying out the container.
    // The child applet should return its full desired size, whereas
    //  the parent in the browser can be small.
    public synchronized Dimension preferredSize()
    {
        if(this != parentApp)
            return new Dimension(desiredW, desiredH);
        else
            return new Dimension(10,10);
    }

}
```

IA Jim:Desktop Folder:thesiscode:CarLayout.java **Page: 1**
Friday, May 23, 1997 / 6:44 AM

```
// CarLayout.java
// This class subclasses FlowLayout and overrides
//  layoutContainer() so we can custom-place components
//  within a frame.


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

public class CarLayout extends FlowLayout
{
    // Initializes the layout manager.
    public CarLayout()
    {
        super();
    }

    // This version makes components lay themselves
    //  out.  It can be extended by subclassing other
    //  components and adding extra if(curComp instanceof ...)
    //  statements.
    public void layoutContainer(Container target)
    {
        int             nmembers = target.countComponents();
        int             index;
        Component       curComp = null;

        for(index = 0; index < nmembers; index++)
        {
            curComp = target.getComponent(index);
            if(curComp != null)
            {
                if(curComp instanceof ImageButton)
                    ((ImageButton)curComp).ReshapeButton();
            }
        }
    }
}
```

```
// DashImageFilter.java
// Source for a transparency image filter.

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


class DashImageFilter  extends  RGBImageFilter  {
    public  DashImageFilter()
    {
        //  The  filter's  operation  does  not  depend  on  the
        //  pixel's  location,  so  IndexColorModels  can  be
        //  filtered  directly.
        canFilterIndexColorModel  =  true;
    }

    // This method returns 0 (transparent if alpha bits are 0) for
    //  primarily green colors.
    public  int  filterRGB(int  x,  int  y,  int  rgb)
    {
        if (((rgb & 0xff0000) < 0x300000) && ((rgb & 0xff00) > 0x8000)
            && ((rgb & 0xff) < 0x30))
                return 0;
        else
            return rgb;
    }
}
```

```
// AppFrame.java
// This is the source for the applet window.  It is
//  primarily responsible for screen transitions.

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

public class AppFrame extends Frame {
    CarApplet   parentApp, childApp;    // references the two applets
    Image       offImage;               // references the offscreen buffer
    GoButton    goBtn;                  // references the global go button
    Screen      currentScreen, lastScreen;
    Screen[]    nextScreens;            // an array of all possible next screens
    int         totalNext, desiredNext; // total next screens, and the desired one
    int         frameWidth, frameHeight;  // (probably not necessary)
    boolean     mousedown = false;      // holds mouse state


    // This initializes the frame, the child applet, and the first screen
    // I'll admit that I could abstract a few pieces of this method.
    public AppFrame(CarApplet pApp, CarApplet cApp, String s, int inW, int inH)
    {
        super(s);

        parentApp = pApp;
        childApp = cApp;

        frameWidth = inW;
        frameHeight = inH;

        // sizing the frame
        // picking my layout manager (null does not work well)
        resize(inW, inH);
        setResizable(false);
        setLayout(new CarLayout());

        // creating an offscreen image
        offImage = pApp.createImage(inW, inH);
        if(offImage == null)
            System.out.println("Null offscreen image.");

        // initializing the child applet
        cApp.SetStandalone(true);
        cApp.SetParentApp(pApp);
        cApp.SetFrame(this);

        cApp.init();
        cApp.start();
        add(cApp);
        cApp.reshape(0, 0, inW, inH);
        cApp.setLayout(new CarLayout());
        pack();

        // initializing the array of next screens
        totalNext = 0;
        nextScreens = new Screen[1];
        desiredNext = -1;
        lastScreen = null;
        currentScreen = null;
```

```
        // allocate the go button
        goBtn = new GoButton(cApp);

        // add first screen to list of next screens
        // load its items
        // choose first screen as next screen
        // transition to next screen
        IN_01   firstScreen = new IN_01(this);
        AddNextScreen(firstScreen);
        PreloadNextScreens();
        PerformScreenTransition(GoButton.PUSH_FWD);
    }

    public CarApplet GetParentApplet()
    {
        return parentApp;
    }

    public CarApplet GetChildApplet()
    {
        return childApp;
    }

    public int GetFrameHeight()
    {
        return frameHeight;
    }

    public int GetFrameWidth()
    {
        return frameWidth;
    }

    public Screen GetLastScreen()
    {
        return lastScreen;
    }

    public Screen GetCurrentScreen()
    {
        return currentScreen;
    }

    public Screen[] GetNextScreens()
    {
        return nextScreens;
    }

    // Starts preloading the images for all possible next screens.
    public void PreloadNextScreens()
    {
        int index;

        for(index = 0; index < totalNext; index++)
        {
            // If we've already visited this screen,
            //  we need to replace a few values
            if(nextScreens[index].AlreadyVisited())
            {
                nextScreens[index].SetScreenFrame(this);
                nextScreens[index].PresetCharacteristics();
            }
            nextScreens[index].BeginPreloading();
        }
    }

    // Checks if any of images for the possible next screens
    //  is still preloading.
    public boolean AnyBusyPreloading()
    {
        boolean         temp = true;
        int             index;

        for(index = 0; index < totalNext; index++)
        {
            if(nextScreens[index] != null)
            {
                temp = temp & nextScreens[index].FinishedPreloading();
            }
            else
            {
                temp = false;
                break;
            }
        }
        // temp is now true iff all have loaded
        //  thus !temp is false iff all have loaded

        return !temp;
    }

    // This looks for a particular screen reference in the
    //  array of possible next screens and sets the desiredNext
    //  index to it.  If it can't find the screen, it returns
    //  false.
    public boolean SetNextScreen(Screen nextS)
    {
        int index;
        int tempSave = desiredNext;

        desiredNext = -1;
        for(index = 0; index < totalNext; index++)
        {
            if(nextScreens[index] == nextS)
            {
                desiredNext = index;
                break;
            }
        }

        // If it's not here, revert desiredNext to its old state
        //  and return false.
        if(desiredNext < 0)
        {
            desiredNext = tempSave;
            return false;
        }
        else
            return true;
    }

    // This adds a screen to the list of possible next screens.
    public void AddNextScreen(Screen nextS)
```

```
    {
        if(nextS == null)
            return;

        if(totalNext == 0)
        {
            desiredNext = 0;    // If only one screen, it's not necessary
                                //  to use SetNextScreen().
            nextScreens[0] = nextS;
        }
        else
        {
            Screen tempList[] = new Screen[totalNext + 1];

            System.arraycopy(nextScreens, 0, tempList, 0, totalNext);
            tempList[totalNext] = nextS;

            nextScreens = tempList;
        }

        totalNext++;
    }

    // Removes all possible next screens and resets the list.
    public void PurgeNextScreens()
    {
        int     index;

        for(index = 0; index < totalNext; index++)
        {
            if(nextScreens[index] != currentScreen)
                nextScreens[index].CleanUpAfter();
        }

        totalNext = 0;
        nextScreens = new Screen[1];
    }

    // !!Important Method!! This method transitions screens.
    public void PerformScreenTransition(int dir)
    {
        // remove currently visible screen (and its items)
        if(currentScreen != null)
            currentScreen.CleanUpAfter();
        lastScreen = currentScreen;

        // If we're going forward, wait if we're still
        //  preloading the images.
        if(dir == GoButton.PUSH_FWD)
        {
            if(!nextScreens[desiredNext].FinishedPreloading())
                nextScreens[desiredNext].WaitForImages();
            currentScreen = nextScreens[desiredNext];

            if(lastScreen != null)
                lastScreen.SetNextScreen(currentScreen);
            currentScreen.SetLastScreen(lastScreen);
        }
        else
        {
            // Do reloading of last screen here.
            Screen  oneBack = currentScreen.GetLastScreen();
            if(oneBack != null)
            {
                oneBack.SetScreenFrame(this);
                oneBack.PresetCharacteristics();
                oneBack.BeginPreloading();
                if(!oneBack.FinishedPreloading())
                    oneBack.WaitForImages();
                currentScreen = oneBack;
            }
            // If no previous screen, reload current screen.  (In case of errors...)
            else
            {
                currentScreen.SetScreenFrame(this);
                currentScreen.PresetCharacteristics();
                currentScreen.BeginPreloading();
                if(!currentScreen.FinishedPreloading())
                    currentScreen.WaitForImages();
            }

            // Don't revise the "last screen" field...  We want to preserve ordering.
        }

        // Reset the list of possible next screens.
        PurgeNextScreens();

        // add next screen (and its items)
        currentScreen.DisplayAndReady();
    }

    // All components other than buttons belong to
    //  the specific screens, so the current screen
    //  is allowed to intercept actions before passing
    //  them up the line.
    public boolean action(Event evt, Object what)
    {
        if(evt.target instanceof ScreenButton)
        {
            if(currentScreen != null)
            {
                if(currentScreen.action(evt, what))
                    return true;
            }
        }
        return super.action(evt, what);
    }

    // If we have a mouseUp event in the frame,
    //  check to see if it changed the status of
    //  a "MyButton" instance before sending it up
    //  the line.
    public boolean mouseUp(Event evt, int x, int y)
    {
        boolean     handled = false;

        mousedown = false;
        if(currentScreen != null)
        {
            Point       loc = new Point(x, y);
            if(currentScreen.UseGoButton())
                handled = UpdateGoIfChanged(loc);
            if(!handled)
                handled = currentScreen.UpdateMyButtonsIfChanged(mousedown, loc);
```

```
        }

        return handled;
    }

    // If we have a mouseDown event in the frame,
    //  check to see if it changed the status of
    //  a "MyButton" instance before sending it up
    //  the line.
    public boolean mouseDown(Event evt, int x, int y)
    {
        boolean     handled = false;

        mousedown = true;
        if(currentScreen != null)
        {
            Point       loc = new Point(x, y);
            if(currentScreen.UseGoButton())
                handled = UpdateGoIfChanged(loc);
            if(!handled)
                handled = currentScreen.UpdateMyButtonsIfChanged(mousedown, loc);
        }

        return handled;
    }

    // If we have a mouseMove event in the frame,
    //  check to see if it changed the status of
    //  a "MyButton" instance before sending it up
    //  the line.
    public boolean mouseMove(Event evt, int x, int y)
    {
        boolean     handled = false;

        if(currentScreen != null)
        {
            Point       loc = new Point(x, y);
            if(currentScreen.UseGoButton())
                handled = UpdateGoIfChanged(loc);
            if(!handled)
                handled = currentScreen.UpdateMyButtonsIfChanged(mousedown, loc);
        }

        return handled;
    }

    // This is an abstraction to ask the go button if a given
    //  point and mouse state changed its current state.
    public boolean UpdateGoIfChanged(Point pt)
    {
        if(goBtn != null)
        {
            if(goBtn.ChangedState(pt, mousedown))
            {
                goBtn.UpdateButton();
                return true;
            }
        }
        return false;
    }

    public GoButton GetGoButton()
    {
        return goBtn;
    }

    // This is an abstraction to ask the parent applet to grab
    //  an image.  The child applet could do it if I changed
    //  its allocation slightly...  (A future improvement.)
    public Image GrabImage(String s)
    {
        return parentApp.getImage(parentApp.getDocumentBase(), s);
    }


    // The following list handling routines will probably
    // be placed in a global toolbox.  They were originally
    // put here since we'd usually only have one copy of a
    // frame at a time.

    public Point[] AddPointToList(Point[] existingList, Point newPt)
    {
        Point[]     tempPList;
        if(existingList != null) {
            tempPList = new Point[existingList.length + 1];
            System.arraycopy(existingList, 0, tempPList, 0, existingList.length);
            tempPList[existingList.length] = newPt;
        }
        else {
            tempPList = new Point[1];
            tempPList[0] = newPt;
        }
        return tempPList;
    }

    public Image[] AddImageToList(Image[] existingList, Image newImage)
    {
        Image[]     tempIList;
        if(existingList != null) {
            tempIList = new Image[existingList.length + 1];
            System.arraycopy(existingList, 0, tempIList, 0, existingList.length);
            tempIList[existingList.length] = newImage;
        }
        else {
            tempIList = new Image[1];
            tempIList[0] = newImage;
        }
        return tempIList;
    }

    public ScreenButton[] AddScreenButtonToList(ScreenButton[] existingList,
                                                ScreenButton newButton)
    {
        ScreenButton[]      tempSBList;
        if(existingList != null) {
            tempSBList = new ScreenButton[existingList.length + 1];
            System.arraycopy(existingList, 0, tempSBList, 0, existingList.length);
            tempSBList[existingList.length] = newButton;
        }
        else {
            tempSBList = new ScreenButton[1];
            tempSBList[0] = newButton;
        }
        return tempSBList;
```

```
    }


    // This is needed to allow the window to close by clicking the
    //  close box.  Eventually, we will want to save state before
    //  closing the window.
    public boolean handleEvent(Event e) {
        // Window Destroy event
        if (e.id == Event.WINDOW_DESTROY) {
            if(parentApp != null)
                parentApp.SetFrame(null);
            childApp = null;
            dispose();
            return true;
        }
        return super.handleEvent(e);
    }

    public synchronized Dimension preferredSize()
    {
        return new Dimension(frameWidth, frameHeight);
    }
}
```

```java
// Screen.java
// This is the superclass of every screen instance,
// containing common code for all screens.

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

public class Screen extends Object {
    AppFrame            theFrame = null;
    MediaTracker        theTracker = null;
    Screen              lastScreen = null;
    Screen              nextScreen = null;

    ScreenButton[]      buttons = null;     // Array of ScreenButtons
    int                 totalButtons = 0;   // (unnecessary)

    MyButtonSet[]       myButtons = null;   // Array of MyButtonSets

    boolean             useGoBtn = true;    // Do we use the Go button?
    boolean             allowFwd = true;    // Fwd enabled?
    boolean             allowRev = true;    // Rev enabled?

    Image[]             images = null;      // Array of images
    Point[]             imageCoords = null; // Origins of the images
    int                 totalImages = 0;    // (unnecessary)

    boolean             visited = false;    // Visited before?
    boolean             proceed = false;    // User allowed to proceed?


    // Don't bother doing anything if we don't have a
    //  frame reference.
    public Screen()
    {
    }

    // With a valid frame, initialize everything.
    public Screen(AppFrame f)
    {
        if(f == null)
            return;
        InitAll(f);
    }

    // Abstraction to paint all of the MyButtonSets.
    public void PaintAllMyButtonSets(Graphics g)
    {
        int     index;

        if(myButtons != null)
        {
            for(index = 0; index < myButtons.length; index++)
                myButtons[index].PaintAllButtons(g);
        }
    }

    // Abstraction to forward events to buttons in all sets.
    // An improved method could break from the loop on a change.
    public boolean UpdateMyButtonsIfChanged(boolean mousedown, Point loc)
```

```java
    {
        int         index;
        boolean     oneChanged = false;

        if(myButtons != null)
        {
            for(index = 0; index < myButtons.length; index++)
            {
                oneChanged |= myButtons[index].AnyChangedState(loc, mousedown);
            }
        }
        return oneChanged;
    }

    // Adds a MyButtonSet to the list.
    public void AddMyButtonSet(MyButtonSet btnSet)
    {
        int     index;

        if(btnSet != null)
        {
            if(myButtons == null)
            {
                myButtons = new MyButtonSet[1];
                myButtons[0] = btnSet;
            }
            else
            {
                MyButtonSet     tempList[] = new MyButtonSet[myButtons.length + 1];
                System.arraycopy(myButtons, 0, tempList, 0, myButtons.length);
                tempList[myButtons.length] = btnSet;
                myButtons = tempList;
            }
        }
    }


    // Flushes the images and references of all MyButtons
    //  in the screen.
    public void RemoveAllMyButtonSets()
    {
        int     index;

        if(myButtons != null)
        {
            for(index = 0; index < myButtons.length; index++)
            {
                myButtons[index].FlushMyButtonSet();
            }
        }

        myButtons = null;
    }

    public boolean AlreadyVisited()
    {
        return visited;
    }

    public void SetVisited(boolean v)
    {
        visited = v;
    }

    public void AllowProceed(boolean p)
    {
        proceed = p;
        SetGoButton();
    }


    // When initializing, reset fields to their defaults
    //  and add images to the frame for preloading...
    public void InitAll(AppFrame f)
    {
        if(f == null)
            return;
        theFrame = f;

        PresetCharacteristics();
        ResetFields();

        visited = true;
    }

    // Override this in subclasses if desired.
    public void ResetFields()
    {
    }

    // This should always be overridden for adding screen images, etc.
    public void PresetCharacteristics()
    {
    }


    public void SetLastScreen(Screen s)
    {
        lastScreen = s;
    }

    public Screen GetLastScreen()
    {
        return lastScreen;
    }

    public void SetNextScreen(Screen s)
    {
        nextScreen = s;
    }

    public Screen GetNextScreen()
    {
        return nextScreen;
    }

    public void SetScreenFrame(AppFrame f)
    {
        theFrame = f;
    }

    public AppFrame GetScreenFrame()
    {
        return theFrame;
```

```
    }

    public boolean UseGoButton()
    {
        return useGoBtn;
    }

    // I think this is a vestigial method.  It can probably disappear.
    public void InitImageList(int listLength)
    {
        totalImages = 0;
        images = null;
        imageCoords = null;
    }

    public boolean OkayToProceed()
    {
        return proceed;
    }


    // Add an image to the image list.
    public void AddImage(Image anImage, Point origin)
    {
        if(anImage != null)
        {
            images = theFrame.AddImageToList(images, anImage);
            imageCoords = theFrame.AddPointToList(imageCoords, origin);
            totalImages++;
        }
    }

    // Add a ScreenButton to the list.
    public void AddScreenButton(ScreenButton theButton, Point coords)
    {
        if((theButton != null) && (theFrame != null))
        {
            buttons = theFrame.AddScreenButtonToList(buttons, theButton);
            totalButtons++;
            if(coords != null)
            {
                theButton.SetPosition(coords);
            }
        }
    }


    // The addition/removal routines should also
    //  eventually call some other abstraction, just
    //  in case the implementation changes.

    // Add all of the ScreenButtons to the child applet for display.
    public void AddAllButtons()
    {
        int         index;
        CarApplet   childApplet = null;

        if(theFrame != null)
        {
            childApplet = theFrame.GetChildApplet();
            if(childApplet != null)
            {
                for(index = 0; index < totalButtons; index++)
                {
                    childApplet.add(buttons[index]);
                }
            }
        }
    }

    // Remove all of the ScreenButtons from the child applet.
    public void RemoveAllButtons()
    {
        int         index;
        CarApplet   childApplet = null;

        if(theFrame != null)
        {
            childApplet = theFrame.GetChildApplet();
            if(childApplet != null)
            {
                for(index = 0; index < totalButtons; index++)
                {
                    childApplet.remove(buttons[index]);
                }
            }
        }
    }

    // Add a single ScreenButton to the child applet.
    public void AddSingleButton(ScreenButton theButton)
    {
        int         index;
        CarApplet   childApplet = null;

        if(theFrame != null)
        {
            childApplet = theFrame.GetChildApplet();
            if(childApplet != null)
            {
                for(index = 0; index < totalButtons; index++)
                {
                    if(theButton == buttons[index])
                    {
                        childApplet.add(buttons[index]);
                        break;
                    }
                }
            }
        }
    }

    // Remove a single ScreenButton from the child applet.
    public void RemoveSingleButton(ScreenButton theButton)
    {
        CarApplet   childApplet = null;

        if(theFrame != null)
        {
            childApplet = theFrame.GetChildApplet();
            if(childApplet != null)
                childApplet.remove(theButton);
        }
    }
```

```
    // Begin preloading all of the images with a
    //  new MediaTracker.
    public void BeginPreloading()
    {
        int index;

        theTracker = new MediaTracker(theFrame);

        for(index = 0; index < totalImages; index++)
        {
            theTracker.addImage(images[index], 0);
        }

        theTracker.statusID(0, true);
    }

    public MediaTracker GetTracker()
    {
        return theTracker;
    }

    // Are we done preloading images?
    public boolean FinishedPreloading()
    {
        if(theTracker != null)
            return theTracker.checkID(0, true);
        else
            return false;
    }

    // Wait for all images to finish preloading.
    //  Note: We will want to do something with the
    //        exception eventually.
    public void WaitForImages()
    {
        try {
            theTracker.waitForID(0);
        }
        catch(InterruptedException exc)
        {
        }
    }


    // This makes the last few changes to
    //  the screen that had to wait until the
    //  frame transitioned to it.
    // Override ONLY if appropriate.
    public void DisplayAndReady()
    {
        CarApplet   childApp = null;

        if(theFrame != null)
        {
            SetGoButton();

            AddAllButtons();

            childApp = theFrame.GetChildApplet();
            if(childApp != null)
            {
                childApp.layout();
                theFrame.repaint();
            }

            AddAllNextScreens();
            theFrame.PreloadNextScreens();
        }
    }

    // This allocates and adds all possible next
    //  screens to the frame.
    // Override in all screens.
    public void AddAllNextScreens()
    {
    }

    // This is an abstraction for adding possible
    //  next screens to the frame.
    public void AddNextScreen(Screen s)
    {
        if(nextScreen != null)
        {
            if(s != nextScreen)
            {
                if(CompareToPrevNext((s.getClass()).getName()))
                {
                    theFrame.AddNextScreen(nextScreen);
                    s.CleanUpAfter();
                    return;
                }
            }
        }
        theFrame.AddNextScreen(s);
    }


    // Only a screen can set the go button with respect
    //  to its enable flags.
    private void SetGoButton()
    {
        GoButton    goBtn = theFrame.GetGoButton();
        if(goBtn != null)
        {
            goBtn.Show(useGoBtn);
            if(useGoBtn)
            {
                if(proceed)
                {
                    goBtn.EnableRev(allowRev);
                    goBtn.EnableFwd(allowFwd);
                }
                else
                    goBtn.EnableButton(false);
            }
        }
    }


    // Paint all of the screen's images.
    public void PaintImages(Graphics g)
    {
        int index;
```

```
        for(index = 0; index < totalImages; index++)
        {
            g.drawImage(images[index], imageCoords[index].x,
                        imageCoords[index].y, theFrame);
        }
    }

    // Clean up all references and be sure to flush
    //  ALL images.
    public void CleanUpAfter()
    {
        int index;

        for(index = 0; index < totalImages; index++)
        {
            images[index].flush();
        }

        // Clean up ScreenButtons.
        RemoveAllButtons();
        for(index = 0; index < totalButtons; index++)
        {
            buttons[index].CleanUpImages();
        }

        // Clean up MyButtonSets.
        RemoveAllMyButtonSets();

        theFrame = null;
        theTracker = null;
        images = null;
        imageCoords = null;
        buttons = null;
        totalImages = 0;
        totalButtons = 0;

        // Force a garbage collection.
        System.gc();
    }


    // This will compare the desired next screen to the previously
    // visited next screen.  If they're of the same type, we want
    // to know so we don't make another one.
    public boolean CompareToPrevNext(String s)
    {
        if(nextScreen != null)
        {
            System.out.println((nextScreen.getClass()).getName());
            System.out.println(s);
            if(s.equals((nextScreen.getClass()).getName()))
            {
                System.out.println("Match.  We've seen this before.");

                return true;
            }
        }
        return false;
    }

    // This can get called for any action (not just ScreenButton events).
    // evt.target gives us the particular button.
    // Override action() to respond to the various buttons.
    public boolean action( Event evt, Object what )
    {
        return false;
    }
}
```

IA Jim:Desktop Folder:thesiscode:IN_01.java **Page: 1**
Friday, May 23, 1997 / 6:49 AM

```
// IN_01.java
// This is an example screen.  It loads a background image,
//  creates two sets of buttons, disables the reverse option
//  of the Go button, and allocates one screen as the next screen.

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

public class IN_01 extends Screen {
    // Just call super().  We don't want any special initializations.
    public IN_01(AppFrame f)
    {
        super(f);
    }

    // Don't worry about adding MyButtonSets here, since
    //  they're only given events if this screen is the "current"
    //  screen.
    // ScreenButtons, however, should be added in AddAllNextScreens().
    public void PresetCharacteristics()
    {
        // Get and add the background image.
        Image       screenImg = theFrame.GrabImage("Screens/IN_01.jpg");

        AddImage(screenImg, new Point(0,0));

        // Set flags for the Go button.
        proceed = true;
        allowRev = false;

        // Make an array of points for the origins of 4 radio buttons.
        Point[]     origins = new Point[4];
        origins[0] = new Point(100, 100);
        origins[1] = new Point(100, 180);
        origins[2] = new Point(100, 260);
        origins[3] = new Point(100, 340);

        // Add a set of 4 radio buttons with the above origins.
        AddMyButtonSet(new RadioButtonSet(theFrame.GetChildApplet(),
                                          screenImg, 4, origins));

        // Make an array of points for the origins of 4 checkbox buttons.
        Point[]     origins2 = new Point[4];
        origins2[0] = new Point(300, 100);
        origins2[1] = new Point(300, 180);
        origins2[2] = new Point(300, 260);
        origins2[3] = new Point(300, 340);

        // Add a set of 4 checkbox buttons with the above origins.
        AddMyButtonSet(new CheckBoxButtonSet(theFrame.GetChildApplet(),
                                             screenImg, 4, origins2));
    }


    // Add the next screens to the frame for preloading.
    // Add ScreenButtons here, since they're added immediately
    //  to the child applet.
    public void AddAllNextScreens()
    {
```

IA Jim:Desktop Folder:thesiscode:IN_01.java **Page: 2**
Friday, May 23, 1997 / 6:50 AM

```
        if(!CompareToPrevNext(new String("IN_02")))
            AddNextScreen(new IN_02(theFrame));
        else
            AddNextScreen(nextScreen);
    }
}
```

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

public class MyButton extends Object {
    Image                       background; // background region the
                                            // same size as the button
    Image                       images[];   // the array of button images

    Point                       origin;     // the origin of the button
    Dimension                   size;       // the button's size

    DashImageFilter             filter;     // the transparency filter
    CropImageFilter             cropFilter; // the crop filter for the background
    CarApplet                   theApp;
    AppFrame                    theFrame;
    int                         state;      // the current state
    // savedState is used whenever another button is being
    //  pushed and others need to temporarily submit to it.
    // If the button is actually pushed, state will not revert
    //  to savedState, but will remain the new state.
    int                         savedState;
    boolean                     inTemp = false;
    // Type: rect, round, rounded rect.  RECT is default.
    int                         type = RECT;

    boolean                     show = true;
    boolean                     enabled = true;

    MyButton                    companions[] = null;     // The button's
                                                          //  companions
    int                         totalImages;    // (unnecessary)

    public static final int     RECT = 0;       // Constants for the shapes
    public static final int     ROUND = 1;
    public static final int     ROUNDRECT = 2;

    public static final int     MSG_NULL = 0;   // Constants for message types
    public static final int     MSG_PUSHBEGIN = 1;
    public static final int     MSG_STATESET = 2;
    public static final int     MSG_COMMIT = 3;
    public static final int     MSG_ABORT = 4;

    // room for more message types


    // If we do not get any information, don't bother initializing anything.
    public MyButton()
    {
    }

    // If we get the information we need, initialize everything.
    public MyButton(CarApplet inApp, Image screenImage, int numImages,
                        Point org, Dimension s)
    {
        theApp = inApp;
        if(inApp != null)
        {
            theFrame = inApp.GetFrame();
            if(theFrame != null)
```

```
            {
                totalImages = numImages;
                origin = org;
                size = s;

                if(screenImage == null)
                    screenImage = theFrame.GrabImage("Buttons/template.jpg");
                if(InitFilters());
                    InitImages(screenImage);
            }
        }
    }

    // Reset the button to a base state.
    public void ResetButton()
    {
        inTemp = false;
        state = 0;
    }

    // This might be useful for radio buttons that
    //  temporarily turn off while another is pushed.
    // (If the push is aborted, we can easily switch back.)
    //
    // Actually, it's useful for any aborted push.
    public void BeginTemp(int tempState)
    {
        savedState = state;
        state = tempState;

        inTemp = true;
    }

    // If other action did not commit, revert to old state.
    public void EndTemp(boolean commit)
    {
        if(!commit)
            state = savedState;

        inTemp = false;
    }

    // Should be used only to coerce a button to an initial state.
    // We notify companions to simplify initialization of a set.
    //  This way, if doing "radio buttons," we only need to set one.
    public void SetState(int s)
    {
        state = s;
        NotifyCompanions(MSG_STATESET, new Integer(s));
    }

    public int GetState()
    {
        return state;
    }


    // Tests if a given button is a companion.
    public boolean IsCompanion(MyButton btn)
    {
        boolean     test = false;
        int         index;
        if(companions != null)
        {
            for(index = 0; index < companions.length; index++)
            {
                if(companions[index] == btn)
                    test = true;
            }
        }

        return test;
    }


    // Makes another button a companion.
    public void AddCompanion(MyButton btn)
    {
        if(companions == null)
        {
            companions = new MyButton[1];
            companions[0] = btn;
        }
        else
        {
            MyButton    tempList[] = new MyButton[companions.length + 1];
            System.arraycopy(companions, 0, tempList, 0, companions.length);
            tempList[companions.length] = btn;
            companions = tempList;
        }
    }

    // Removes a button from the companion list.
    public void RemoveCompanion(MyButton btn)
    {
        // Assumes companion is only in here once.
        int         index;
        int         where = 0;
        boolean     inThere = false;
        MyButton    tempList[];

        if(companions != null)
        {
            for(index = 0; index < companions.length; index++)
            {
                if(companions[index] == btn)
                {
                    where = index;
                    inThere = true;
                }
            }

            if(inThere)
            {
                tempList = new MyButton[companions.length - 1];
                System.arraycopy(companions, 0, tempList, 0, where);
                System.arraycopy(companions, where + 1, tempList, where,
                                 tempList.length - where);
            }
        }
    }

    // Resets the companion list, removing all companions at once.
```

```
    public void RemoveAllCompanions()
    {
        companions = null;
    }

    // Sends a message to all companions.
    public void NotifyCompanions(int msgType, Object data)
    {
        int     index;

        if(companions != null)
        {
            for(index = 0; index < companions.length; index++)
            {
                if(companions[index] != null)
                    companions[index].HandleMessage(this, msgType, data);
            }
        }
    }

    // Sends a message to a single companion.
    public void NotifyOneCompanion(MyButton btn, int msgType, Object data)
    {
        btn.HandleMessage(this, msgType, data);
    }


    // Override as necessary.
    // The data can be any convenient data object for the specific button type.
    public void HandleMessage(MyButton btn, int msgType, Object data)
    {
    }

    public int GetType()
    {
        return type;
    }

    public void SetType(int t)
    {
        type = t;
    }

    public void Show()
    {
        show = true;
    }

    public void Show(boolean s)
    {
        show = s;
    }

    public void Hide()
    {
        show = false;
    }

    public boolean IsShown()
    {
        return show;
    }
    public void EnableButton(boolean en)
    {
        enabled = en;
    }

    public boolean Enabled()
    {
        return enabled;
    }

    public void SetApplet(CarApplet inApp)
    {
        theApp = inApp;
    }

    // Return a reference to one of the button's state images.
    public Image GetImage(int index)
    {
        if((index >= 0) && (index < images.length))
            return images[index];
        else
            return (Image)null;
    }

    // Change/Set one of the button's state images.
    public void SetImage(int index, Image img)
    {
        if((index >= 0) && (index < images.length))
            images[index] = img;
    }


    // Override as necessary with each button type.
    public void InsertButtonImages(Image[] tempImages)
    {
    }

    // Flushes all of the button's images.
    public void FlushAll()
    {

        int     index;

        for(index = 0; index < images.length; index++)
        {
            if(images[index] != null)
                images[index].flush();
        }

        if(background != null)
            background.flush();

        SecondaryFlush();
    }

    // Removes references to other objects.
    public void SecondaryFlush()
    {
        RemoveAllCompanions();

        images = null;
```

```
        background = null;
        theApp = null;
        filter = null;
        cropFilter = null;
        origin = null;
        size = null;
    }

    // Initializes both the transparency and the crop filters.
    public boolean InitFilters()
    {
        // We'd probably like an exception or error if not loaded.

        InitDashImageFilter();
        InitCropImageFilter();

        return ((cropFilter != null) && (filter != null));
    }

    public void InitDashImageFilter()
    {
        filter = new DashImageFilter();
    }

    public void InitCropImageFilter()
    {
        cropFilter = new CropImageFilter(origin.x, origin.y,
                            size.width, size.height);
    }


    // A call just to PaintButton() assumes that the background
    // has been already drawn behind the button, as in a full redraw
    // of the display.
    // UpdateButton() should be used whenever one just wants to
    // redraw the button and not the rest of the display.

    // Override as appropriate.
    public void PaintButton(Graphics g)
    {
    }


    // Note that this must change if the offscreen buffer ever moves
    // from AppFrame.
    public void UpdateButton()
    {
        if(!show)
            return;
        if(theApp != null)
        {
            if(theFrame != null)
            {
                if((theFrame.offImage != null) &&
                    (cropFilter != null))
                {
                // The following is not really necessary for the first three
                // states, but we're just being careful...
                    Graphics    g = theFrame.offImage.getGraphics();
                    g.drawImage(background, origin.x, origin.y, theApp);

                    PaintButton(g);
                    // Just copy the rectangular region around the
                    // button from offscreen to onscreen instead
                    // of the whole buffer.  Is this faster?
                    Image   tempImage = theApp.createImage(new
                                FilteredImageSource(
                                    theFrame.offImage.getSource(),
                                    cropFilter));
                    if(tempImage != null)
                        theApp.getGraphics().drawImage(tempImage, origin.x,
                            origin.y, theApp);
                }
                else // If we don't have an offscreen image, paint it onscreen.
                {
                    PaintButton(theApp.getGraphics());
                }
            }
        }
    }

    public Point GetOrigin()
    {
        return origin;
    }

    public Dimension GetSize()
    {
        return size;
    }

    public void SetOrigin(Point org)
    {
        origin = org;
    }

    public void SetSize(Dimension s)
    {
        size = s;
    }

    // This initializes all of the button's state images.
    // Most buttons should only have to override InsertButtonImages().
    public void InitImages(Image screenImage)
    {
        Image       tempImages[] = new Image[totalImages];

        images = new Image[totalImages];

        if((images == null) || (tempImages == null))
            return;

        ForceScreenImageLoad(screenImage);
        MakeBackground(screenImage);
        ForceLoadNewBackground();

        InsertButtonImages(tempImages);
        FilterButtonImages(tempImages);
        ForceLoadAllImages(tempImages);

        FlushTempImages(tempImages);
    }
```

```
    // Runs the state images through the transparency filter.
    public void FilterButtonImages(Image[] tempImages)
    {
        int         index;
        boolean     nullTest = false;

        images = new Image[totalImages];

        for(index = 0; index < totalImages; index++)
        {
            if(tempImages[index] == null)
                nullTest = true;
        }
        if(!nullTest && (images != null))
        {
            for(index = 0; index < totalImages; index++)
            {
                images[index] = theApp.createImage(
                                    new FilteredImageSource(
                                     tempImages[index].getSource(),
                                     filter));
            }
        }
    }

    // Forces all images to load.
    public void ForceLoadAllImages(Image[] tempImages)
    {
        MediaTracker    theTracker;

        if((theFrame != null) && (images != null) && (tempImages != null))
        {
            theTracker = new MediaTracker(theFrame);
            if(theTracker != null)
            {
                int         index;

                for(index = 0; index < images.length; index++)
                {
                    if(tempImages[index] != null)
                        theTracker.addImage(tempImages[index], 0);
                    if(images[index] != null)
                        theTracker.addImage(images[index], 0);
                }

                WaitForImages(theTracker, 0);
            }
        }
    }

    // Waits for all images to load.
    public void WaitForImages(MediaTracker theTracker, int id)
    {
        theTracker.statusID(id, true);
        try {
            theTracker.waitForAll();
        }
        catch(InterruptedException exc)
        {
        }
    }
    // Forces the new background to load.
    public void ForceLoadNewBackground()
    {
        MediaTracker    theTracker;

        if((theFrame != null) && (background != null))
        {
            theTracker = new MediaTracker(theFrame);
            if(theTracker != null)
            {
                theTracker.addImage(background, 0);
                WaitForImages(theTracker, 0);
            }
        }
    }

    // Flushes an array of temporary images.  (Nice tool/abstraction.)
    public void FlushTempImages(Image[] temps)
    {
        int     index;

        if(temps != null)
        {
            for(index = 0; index < temps.length; index++)
            {
                if(temps[index] != null)
                    temps[index].flush();
            }
        }
    }

    // Crops the full background to the region behind the button.
    public void MakeBackground(Image screenImage)
    {
        if((screenImage != null) && (theApp != null))
        {
            background = theApp.createImage(
                            new FilteredImageSource(screenImage.getSource(),
                                cropFilter));
        }
    }

    // Forces the background screen image to load.
    public void ForceScreenImageLoad(Image screenImage)
    {
        MediaTracker    theTracker;

        if((theFrame != null) && (screenImage != null))
        {
            theTracker = new MediaTracker(theFrame);
            if(theTracker != null)
            {
                theTracker.addImage(screenImage, 0);
                WaitForImages(theTracker, 0);
            }
        }
    }


    // Checks if a point is inside a button.
    // Add additional algorithms for other shapes if desired.
```

```
    public boolean InsideButton(Point loc)
    {
        if((loc.x > origin.x) && (loc.x < origin.x + size.width) && (loc.y > origin.y)
            && (loc.y < origin.y + size.height))
                return true;
        else
            return false;
    }

    // Override as appropriate.
    public boolean ChangedState(Point loc, boolean pushed)
    {
        return false;
    }
}
```

/

IA Jim:Desktop Folder:thesiscode:MyButtonSet.java **Page: 1**
Friday, May 23, 1997 / 6:53 AM

```
// MyButtonSet.java
// Combines similar MyButtons into a single set.

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

public class MyButtonSet extends Object {
    int             setSize = 0;        // (unnecessary)
    Point           origins[] = null;   // list of button origins
    MyButton        buttons[] = null;   // list of buttons
    boolean         validSet = false;   // is this set valid?

    // Do not initialize without any information.
    public MyButtonSet()
    {
    }

    // Override to initialize a specific type of MyButtonSet
    public MyButtonSet(CarApplet inApp, Image screenImage,
                            int numBtns, Point[] orgs)
    {
    }


    public boolean IsValidSet()
    {
        return validSet;
    }

    // override to create original button with new items
    public boolean InitOriginalButton(CarApplet inApp)
    {
        return true;
    }

    // override to clone the original buttons 'x' times
    public boolean InitClones()
    {
        return true;
    }

    // Makes all of the buttons in the set companions.
    public void MakeAllCompanions()
    {
        int     index1, index2;

        if(buttons != null)
        {
            for(index1 = 0; index1 < buttons.length; index1++)
            {
                for(index2 = 0; index2 < buttons.length; index2++)
                {
                    if(index1 != index2)
                        buttons[index1].AddCompanion(buttons[index2]);
                }
            }
        }
    }
```

IA Jim:Desktop Folder:thesiscode:MyButtonSet.java **Page: 2**
Friday, May 23, 1997 / 6:53 AM

```
    // Flushes all images from all buttons in the set.
    public void FlushMyButtonSet()
    {
        int     index;

        if(buttons != null)
        {
            buttons[0].FlushAll();

            for(index = 1; index < buttons.length; index++)
            {
                buttons[index].SecondaryFlush();
            }

            buttons = null;
        }
    }

    // An abstraction that passes events to each button in the set.
    public boolean AnyChangedState(Point loc, boolean mousedown)
    {
        int         index;
        boolean     oneChanged = false;

        if(buttons != null)
        {
            for(index = 0; index < buttons.length; index++)
            {
                if(buttons[index].ChangedState(loc, mousedown))
                {
                    buttons[index].UpdateButton();
                    oneChanged = true;
                    break;
                }
            }
        }
        return oneChanged;
    }

    // An abstraction that paints all of the buttons in the set.
    public void PaintAllButtons(Graphics g)
    {
        int     index;

        if((g != null) && (buttons != null))
        {
            for(index = 0; index < buttons.length; index++)
                buttons[index].PaintButton(g);
        }
    }

}
```

```
// GoButton.java
// This is the source for the global Go button.

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

public class GoButton extends MyButton {
    int                 leftLimit = 26;     // Defined HERE
    int                 rightLimit = 56;    // Defined HERE

    boolean             revEnabled = true;  // Defaults are
    boolean             fwdEnabled = true;  //  fully enabled.

    public static final int     DEFAULT = 0;    // Constants for states
    public static final int     OVER_REV = 1;
    public static final int     OVER_FWD = 2;
    public static final int     PUSH_REV = 3;
    public static final int     PUSH_FWD = 4;

    public static final int     ALLOFF = 0;
    public static final int     ALLON = 1;
    public static final int     FLIT_RON = 2;
    public static final int     FPUSH_RON = 3;
    public static final int     RLIT_FON = 4;
    public static final int     RPUSH_FON = 5;

    public static final int     FON_ROFF = 6;
    public static final int     FLIT_ROFF = 7;
    public static final int     FPUSH_ROFF = 8;
    public static final int     RON_FOFF = 9;
    public static final int     RLIT_FOFF = 10;
    public static final int     RPUSH_FOFF = 11;

    public static final int     NUM_IMAGES = 12;

    public GoButton()
    {
        super();
    }

    // Initialize if given an applet reference.
    public GoButton(CarApplet inApp)
    {
        super(inApp, null, NUM_IMAGES, new Point(320, 440),
                new Dimension(80, 44)); // Traits defined HERE.
    }

    // Enable/Disable forward.  If both directions are disabled,
    //  disable the entire button, else, enable the entire button.
    public void EnableFwd(boolean enable)
    {
        fwdEnabled = enable;
        if(!fwdEnabled && !revEnabled)
            enabled = false;
        else
            enabled = true;
    }

    // Enable/Disable reverse.  If both directions are disabled,
```

```
    //  disable the entire button, else, enable the entire button.
    public void EnableRev(boolean enable)
    {
        revEnabled = enable;
        if(!fwdEnabled && !revEnabled)
            enabled = false;
        else
            enabled = true;
    }

    // Enables both directions.
    public void EnableAll(boolean enable)
    {
        EnableFwd(enable);
        EnableRev(enable);
    }

    public boolean GetFwdEnabled()
    {
        return fwdEnabled;
    }

    public boolean GetRevEnabled()
    {
        return revEnabled;
    }


    // Called on initialization.  It loads all of the state images.
    public void InsertButtonImages(Image[] tempImages)
    {
        int     index = 0;

        if(tempImages != null)
        {
            tempImages[index++] = theFrame.GrabImage("Buttons/alloff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/allon.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/flitron.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/fpshron.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rlitfon.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rpshfon.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/fonroff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/flitroff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/fpshroff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/ronfoff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rlitfoff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rpshfoff.jpg");
        }
    }

    // A call just to PaintButton() assumes that the background
    // has been already drawn behind the button, as in a full redraw
    // of the display.
    // UpdateButton() should be used whenever one just wants to
    // redraw the button and not the rest of the display.
    public void PaintButton(Graphics g)
    {
        if(!show)
            return;
        if((g != null) && (theApp != null))
        {
            if(!enabled)
```

```
                g.drawImage(images[ALLOFF], origin.x, origin.y, theApp);
            else
            {
                switch(state)
                {
                    case DEFAULT:
                        if(revEnabled && fwdEnabled)
                            g.drawImage(images[ALLON], origin.x, origin.y, theApp);
                        else if(!revEnabled)
                            g.drawImage(images[FON_ROFF], origin.x, origin.y,
                                        theApp);
                        else if(!fwdEnabled)
                            g.drawImage(images[RON_FOFF], origin.x, origin.y,
                                        theApp);
                        break;
                    case OVER_REV:
                        if(revEnabled)
                        {
                            if(fwdEnabled)
                                g.drawImage(images[RLIT_FON], origin.x,
                                                origin.y, theApp);
                            else
                                g.drawImage(images[RLIT_FOFF], origin.x,
                                                origin.y, theApp);
                        }
                        else if(fwdEnabled)
                                g.drawImage(images[FON_ROFF], origin.x,
                                                origin.y, theApp);
                            // another else shouldn't occur here.
                        break;
                    case OVER_FWD:
                        if(fwdEnabled)
                        {
                            if(revEnabled)
                                g.drawImage(images[FLIT_RON], origin.x,
                                                origin.y, theApp);
                            else
                                g.drawImage(images[FLIT_ROFF], origin.x,
                                                origin.y, theApp);
                        }
                        else if(revEnabled)
                                g.drawImage(images[RON_FOFF], origin.x,
                                                origin.y, theApp);
                            // another else shouldn't occur here.
                        break;
                    case PUSH_REV:
                        if(fwdEnabled)
                            g.drawImage(images[RPUSH_FON], origin.x, origin.y, theApp);
                        else
                            g.drawImage(images[RPUSH_FOFF], origin.x, origin.y, theApp);
                        break;
                    case PUSH_FWD:
                        if(revEnabled)
                            g.drawImage(images[FPUSH_RON], origin.x, origin.y, theApp);
                        else
                            g.drawImage(images[FPUSH_ROFF], origin.x, origin.y, theApp);
                        break;
                }
            }
        }
    }
```

```
    // The logic is greatly simplified since we ignore mouseDrag
    // events.  Right now we don't have to check if we're already
    // pushing the button for forward or reverse, since mouse movements
    // while the mouse button is down are not the usual mouseMove
    // events but are mouseDrag events.
    public boolean ChangedState(Point loc, boolean pushed)
    {
        int     oldState = state;

        if(!enabled)
            return false;

        if(InsideButton(loc))
        {
            // We're inside the button...
            if((loc.x < origin.x + leftLimit))
            {
                if(revEnabled)
                {
                    // This will be referenced once on
                    // a mouseDown event over the REV region.
                    if(pushed)
                        state = PUSH_REV;
                    else
                    {
                        // This section executes whenever we're over
                        // the REV region with the mouse up.

                        // If we were previously pushing REV,
                        // change the screen.
                        if(state == PUSH_REV)
                            NotifyGo(PUSH_REV);
                        state = OVER_REV;
                    }
                }
                // If it's not enabled, just give the default.
                else
                    state = DEFAULT;
            }
            else if((loc.x > origin.x + rightLimit))
            {
                if(fwdEnabled)
                {
                    // This will be referenced once on
                    // a mouseDown event over the FWD region.
                    if(pushed)
                        state = PUSH_FWD;
                    else
                    {
                        // This section executes whenever we're over
                        // the FWD region with the mouse up.

                        // If we were previously pushing FWD,
                        // change the screen.
                        if(state == PUSH_FWD)
                            NotifyGo(PUSH_FWD);
                        state = OVER_FWD;
                    }
                }
                // If it's not enabled, just give the default.
                else
```

```
                        state = DEFAULT;
                }
                // We're not over REV or FWD, so give the default.
                else
                    state = DEFAULT;
            }
            // We're not inside the button, so give the default.
            else
                state = DEFAULT;

            // Return TRUE if the state has changed so the button can be redrawn.
            return (oldState != state);
        }

        // Notify the frame that the user has requested a screen transition.
        private void NotifyGo(int direction)
        {
            if(theApp != null)
            {
                AppFrame    theFrame = theApp.GetFrame();
                if(theFrame != null)
                {
                    switch(direction)
                    {
                        case(PUSH_REV):
                            theFrame.PerformScreenTransition(direction);
                            break;
                        case(PUSH_FWD):
                            theFrame.PerformScreenTransition(direction);
                            break;
                    }
                }
            }
        }

}
```

```
// CheckBoxButton.java
// The source for all custom checkbox buttons.

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

public class CheckBoxButton extends MyButton {
    public static final int     DEFAULT = 0;    // Constants for states
    public static final int     OFF = 0;
    public static final int     OFF_OVER = 1;
    public static final int     OFF_PUSH = 2;
    public static final int     ON = 3;
    public static final int     ON_OVER = 4;
    public static final int     ON_PUSH = 5;

    public static final int     OFF_DISABLED = 6;
    public static final int     ON_DISABLED = 7;

    public static final int     NUM_IMAGES = 8;

    public CheckBoxButton()
    {
        super();
    }


    // Initialize with the specified origin.
    public CheckBoxButton(CarApplet inApp, Image screenImage, Point org)
    {
        // The size needs to be fixed.
        super(inApp, screenImage, NUM_IMAGES, org, new Dimension(40, 39));
        SetType(RECT);
    }

    // Load all of the button's state images.
    public void InsertButtonImages(Image[] tempImages)
    {
        int     index = 0;

        if(tempImages != null)
        {
            tempImages[index++] = theFrame.GrabImage("Buttons/cboff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cboffov.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cboffovp.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cbon.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cbonov.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cbonovp.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cdisoff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/cdison.jpg");
        }
    }

    // A call just to PaintButton() assumes that the background
    // has been already drawn behind the button, as in a full redraw
    // of the display.
    // UpdateButton() should be used whenever one just wants to
    // redraw the button and not the rest of the display.
    public void PaintButton(Graphics g)
    {
```

```
        if(!show)
            return;
        if((g != null) && (theApp != null))
        {
            if(!enabled)
            {
                if(state == ON || state == ON_OVER)
                    g.drawImage(images[ON_DISABLED], origin.x,
                                origin.y, theApp);
                else
                    g.drawImage(images[OFF_DISABLED], origin.x,
                                origin.y, theApp);
            }
            else
            {
                if((state >= 0) && (state < 6))
                    g.drawImage(images[state], origin.x, origin.y, theApp);
                else
                    g.drawImage(images[OFF], origin.x, origin.y, theApp);
            }
        }
    }


    // Decides if the button's state has changed.
    public boolean ChangedState(Point loc, boolean pushed)
    {
        int     oldState = state;

        if(!enabled)
            return false;

        if(InsideButton(loc))
        {
            if(pushed)
            {
                switch(state)
                {
                    case ON_OVER:
                        state = ON_PUSH;
                        break;
                    case OFF_OVER:
                        state = OFF_PUSH;
                        break;
                }
            }
            else
            {
                // if we were pushing it, show new change
                // else show as being over (whether already on or off)
                switch(state)
                {
                    case OFF_PUSH:
                    case ON:
                        state = ON_OVER;
                        break;
                    case ON_PUSH:
                    case OFF:
                        state = OFF_OVER;
                        break;
                }
            }
        }
        else
        {
            // when mouse is let go outside, return to previous state
            if(!pushed)
            {
                switch(state)
                {
                    case ON_PUSH:
                    case ON_OVER:
                        state = ON;
                        break;
                    case OFF_PUSH:
                    case OFF_OVER:
                        state = OFF;
                        break;
                }
            }
        }

        // Return TRUE if the state has changed so the button can be redrawn.
        return (oldState != state);
    }
}
```

```
// CheckBoxButtonSet.java
// Combines checkbox buttons into a set.

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

public class CheckBoxButtonSet extends MyButtonSet {
    // Init the set.
    public CheckBoxButtonSet(CarApplet inApp, Image screenImage,
                             int numBtns, Point[] orgs)
    {
        setSize = numBtns;

        if(orgs != null)
            origins = orgs;
        else
            return;

        buttons = new CheckBoxButton[numBtns];
        if(buttons == null)
            return;

        if(InitOriginalButton(inApp, screenImage))
        {
            if(InitClones(screenImage))
            {
                validSet = true;
                return;
            }
        }

        System.out.println("Null handle while initializing checkbox buttons.");
        if(buttons[0] != null)
            buttons[0].FlushAll();

        origins = null;
        buttons = null;
    }

    // Initialize the original button with its images and filters.
    public boolean InitOriginalButton(CarApplet inApp,
                                      Image screenImage)
    {
        buttons[0] = new CheckBoxButton(inApp, screenImage, origins[0]);

        return (buttons[0] != null);
    }

    // Clone the button and make all of the buttons companions.
    public boolean InitClones(Image screenImage)
    {
        int     index;

        if((buttons[0] != null) && (screenImage != null))
        {
            AppFrame    theFrame = buttons[0].theFrame;
            if(theFrame == null)
                return false;
```

```
            for(index = 1; index < buttons.length; index++)
            {
                buttons[index] = new CheckBoxButton();
                if(buttons[index] == null)
                    return false;

                buttons[index].theApp = buttons[0].theApp;
                buttons[index].theFrame = theFrame;
                buttons[index].origin = origins[index];
                buttons[index].images = buttons[0].images;
                buttons[index].size = buttons[0].size;
                buttons[index].filter = buttons[0].filter;
                buttons[index].state = buttons[index].savedState
                                     = CheckBoxButton.OFF;
                buttons[index].type = MyButton.RECT;

                buttons[index].InitCropImageFilter();
                if(buttons[index].cropFilter == null)
                {
                    buttons[index] = null;
                    break;
                }

                buttons[index].MakeBackground(screenImage);
                buttons[index].ForceLoadNewBackground();
            }

            MakeAllCompanions();
            return true;
        }
        else
            return false;
    }

}
```

```
// RadioButton.java
// This is the source for all custom radio buttons.

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

public class RadioButton extends MyButton {
    public static final int     DEFAULT = 0;    // Constants for states
    public static final int     OFF = 0;
    public static final int     OFF_OVER = 1;
    public static final int     OFF_PUSH = 2;
    public static final int     ON = 3;
    public static final int     ON_OVER = 4;
    public static final int     ON_PUSH = 5;

    public static final int     OFF_DISABLED = 6;
    public static final int     ON_DISABLED = 7;

    public static final int     NUM_IMAGES = 8;

    public RadioButton()
    {
        super();
    }

    // Inits the radio button at the specified origin.
    public RadioButton(CarApplet inApp, Image screenImage,
                       Point org)
    {
        super(inApp, screenImage, NUM_IMAGES, org,
                       new Dimension(40, 39));
        SetType(ROUND);
    }

    // Loads the button's state images.
    public void InsertButtonImages(Image[] tempImages)
    {
        int     index = 0;

        if(tempImages != null)
        {
            tempImages[index++] = theFrame.GrabImage("Buttons/rboff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rboffov.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rboffovp.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rbon.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rbonov.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rbonovp.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rdisoff.jpg");
            tempImages[index++] = theFrame.GrabImage("Buttons/rdison.jpg");
        }
    }


    // The data here is the state of the button sending the message.
    public void HandleMessage(MyButton btn, int msgType, Object data)
    {
        int     prevState = state;

        if(btn != null)
```

```
        {
            if(IsCompanion(btn))
            {
                switch(msgType)
                {
                    case MSG_COMMIT:
                        if(((Integer)data).intValue() == ON)
                            state = OFF;
                        break;
                    case MSG_STATESET:
                        if(data != null)
                        {
                            if(((Integer)data).intValue() == ON)
                                state = OFF;
                        }
                        break;
                }

                if(state != prevState)
                    UpdateButton();
            }
        }
    }


    // A call just to PaintButton() assumes that the background
    // has been already drawn behind the button, as in a full redraw
    // of the display.
    // UpdateButton() should be used whenever one just wants to
    // redraw the button and not the rest of the display.
    public void PaintButton(Graphics g)
    {
        if(!show)
            return;
        if((g != null) && (theApp != null))
        {
            if(!enabled)
            {
                if(state == ON || state == ON_OVER)
                    g.drawImage(images[ON_DISABLED], origin.x,
                                origin.y, theApp);
                else
                    g.drawImage(images[OFF_DISABLED], origin.x,
                                origin.y, theApp);
            }
            else
            {
                if((state >= 0) && (state < 6))
                    g.drawImage(images[state], origin.x, origin.y, theApp);
                else
                    g.drawImage(images[OFF], origin.x, origin.y, theApp);
            }
        }
    }


    // Decides if the state of the radio button has changed.
    public boolean ChangedState(Point loc, boolean pushed)
    {
        int     oldState = state;
        if(!enabled)
            return false;

        if(InsideButton(loc))
        {
            if(pushed)
            {
                switch(state)
                {
                    case ON_OVER:
                        state = ON_PUSH;
                        break;
                    case OFF_OVER:
                        state = OFF_PUSH;
                        break;
                }
            }
            else
            {
                // if we were pushing it, notify companions of commit
                // else, show as being over (whether already on or off)
                switch(state)
                {
                    case ON_PUSH:
                    case OFF_PUSH:
                        NotifyCompanions(MSG_COMMIT, new Integer(ON));
                    case ON:
                        state = ON_OVER;
                        break;

                    case OFF:
                        state = OFF_OVER;
                        break;
                }
            }
        }
        else
        {
            // when mouse is let go outside, return to previous state
            if(!pushed)
            {
                switch(state)
                {
                    case ON_PUSH:
                    case ON_OVER:
                        state = ON;
                        break;
                    case OFF_PUSH:
                    case OFF_OVER:
                        state = OFF;
                        break;
                }
            }
        }

        // Return TRUE if the state has changed so the button can be redrawn.
        return (oldState != state);
    }
}
```

```
// RadioButtonSet.java
// Combines radio buttons into a complete set.

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

public class RadioButtonSet extends MyButtonSet {
    // Init the set.
    public RadioButtonSet(CarApplet inApp, Image screenImage,
                            int numBtns, Point[] orgs)
    {
        setSize = numBtns;

        if(orgs != null)
            origins = orgs;
        else
            return;

        buttons = new RadioButton[numBtns];
        if(buttons == null)
            return;

        if(InitOriginalButton(inApp, screenImage))
        {
            if(InitClones(screenImage))
            {
                validSet = true;
                return;
            }
        }

        System.out.println("Null handle while initializing radio buttons.");
        if(buttons[0] != null)
            buttons[0].FlushAll();

        origins = null;
        buttons = null;
    }

    // Initialize the original button with all of its images and filters.
    public boolean InitOriginalButton(CarApplet inApp,
                                        Image screenImage)
    {
        buttons[0] = new RadioButton(inApp, screenImage, origins[0]);

        return (buttons[0] != null);
    }

    // Clone the button and make all the buttons companions.
    public boolean InitClones(Image screenImage)
    {
        int     index;

        if((buttons[0] != null) && (screenImage != null))
        {
            AppFrame    theFrame = buttons[0].theFrame;
            if(theFrame == null)
                return false;
```

```
            for(index = 1; index < buttons.length; index++)
            {
                buttons[index] = new RadioButton();
                if(buttons[index] == null)
                    return false;

                buttons[index].theApp = buttons[0].theApp;
                buttons[index].theFrame = theFrame;
                buttons[index].origin = origins[index];
                buttons[index].images = buttons[0].images;
                buttons[index].size = buttons[0].size;
                buttons[index].filter = buttons[0].filter;
                buttons[index].state = buttons[index].savedState
                                    = RadioButton.OFF;
                buttons[index].type = MyButton.ROUND;

                buttons[index].InitCropImageFilter();
                if(buttons[index].cropFilter == null)
                {
                    buttons[index] = null;
                    break;
                }

                buttons[index].MakeBackground(screenImage);
                buttons[index].ForceLoadNewBackground();
            }

            MakeAllCompanions();
            return true;
        }
        else
            return false;
    }

}
```

```
// Border.java
// Third-party library class for 3-D borders.

/**
 * Copyright(c) 1997 DTAI, Incorporated (http://www.dtai.com)
 *
 *                         All rights reserved
 *
 * Permission to use, copy, modify and distribute this material for
 * any purpose and without fee is hereby granted, provided that the
 * above copyright notice and this permission notice appear in all
 * copies, and that the name of DTAI, Incorporated not be used in
 * advertising or publicity pertaining to this material without the
 * specific, prior written permission of an authorized representative of
 * DTAI, Incorporated.
 *
 * DTAI, INCORPORATED MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES,
 * EXPRESS OR IMPLIED, WITH RESPECT TO THE SOFTWARE, INCLUDING, BUT
 * NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS FOR ANY PARTICULAR PURPOSE, AND THE WARRANTY AGAINST
 * INFRINGEMENT OF PATENTS OR OTHER INTELLECTUAL PROPERTY RIGHTS.  THE
 * SOFTWARE IS PROVIDED "AS IS", AND IN NO EVENT SHALL DTAI, INCORPORATED OR
 * ANY OF ITS AFFILIATES BE LIABLE FOR ANY DAMAGES, INCLUDING ANY
 * LOST PROFITS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES RELATING
 * TO THE SOFTWARE.
 */

import java.awt.*;
import java.awt.image.*;
import java.net.*;

/**
 * Border - draws a configurable border
 *
 * @author  DTAI, Incorporated
 */

public class Border implements Cloneable {

    public static final int NONE = 0;
    public static final int LINE = 1;
    public static final int THREED_IN = 2;
    public static final int THREED_OUT = 3;
    public static final int ETCHED_IN = 4;
    public static final int EMBOSSED_OUT = 5;
    public static final int ROUND_RECT = 6;

    private int type = NONE;
    private Color border = null;
    private int topMargin = 0;
    private int leftMargin = 0;
    private int bottomMargin = 0;
    private int rightMargin = 0;
    private int minMargin = 0;
    private int maxMargin = 0;
    private int borderThickness = 0;

    private static final double BRIGHTER_FACTOR = 0.8;
    private static final double DARKER_FACTOR = 0.65;

    /**
     * Returns a brighter version of this color.
```

```
     * Replaces the "awt.Color" function brighter to use, in my opinion,
     * a better factor (awt used FACTOR = 0.7)
     *
     * @param color     the color to which to apply the factor
     * @return          the new, brighter color
     */
    public static Color brighter( Color color ) {
        Color newcolor = new Color(Math.min((int)(color.getRed()  *(1/BRIGHTER_FACTOR)), 255
             Math.min((int)(color.getGreen()*(1/BRIGHTER_FACTOR)), 255),
             Math.min((int)(color.getBlue() *(1/BRIGHTER_FACTOR)), 255));
        if ( newcolor.equals( color ) ) {
            return Color.white;
        }
        return newcolor;
    }

    /**
     * Returns a darker version of this color.
     * Replaces the "awt.Color" function brighter to use, in my opinion,
     * a better factor (awt used FACTOR = 0.7)
     *
     * @param color     the color to which to apply the factor
     * @return          the new, darker color
     */
    public static Color darker( Color color ) {
        Color newcolor = new Color(Math.max((int)(color.getRed()  *DARKER_FACTOR), 0),
             Math.max((int)(color.getGreen()*DARKER_FACTOR), 0),
             Math.max((int)(color.getBlue() *DARKER_FACTOR), 0));
        if ( newcolor.equals( color ) ) {
            return Color.black;
        }
        return newcolor;
    }

    /**
     * Returns a clone of this Border
     *
     * @return      the new, identical Border
     */
    public Object clone() {
        Border newborder = new Border();
        newborder.type = type;
        newborder.border = border;
        newborder.topMargin = topMargin;
        newborder.leftMargin = leftMargin;
        newborder.bottomMargin = bottomMargin;
        newborder.rightMargin = rightMargin;
        newborder.minMargin = minMargin;
        newborder.maxMargin = maxMargin;
        newborder.borderThickness = borderThickness;
        return newborder;
    }

    /**
     * sets all margins and thicknesses to zero
     */
    public void setNoInsets() {
        topMargin = 0;
        leftMargin = 0;
        bottomMargin = 0;
        rightMargin = 0;
        minMargin = 0;
        maxMargin = 0;
        borderThickness = 0;
    }

    /**
     * Returns the border type (e.g., Border.LINE)
     *
     * @return      the border type id
     */
    public int getType() {
        return type;
    }

    /**
     * Sets the border type (e.g., to Border.LINE)
     *
     * @param type     the border type
     */
    public synchronized void setType( int type ) {
        this.type = type;
    }

    /**
     * Returns the current border color.  If null (the default), a border color is
     * dynamically derived from the current background (passed to the "paint" function).
     *
     * @return     the current Color value for the border
     */
    public Color getBorder() {
        return border;
    }

    /**
     * Sets the current border color.  If null (the default), a border color is
     * dynamically derived from the current background (passed to the "paint" function).
     *
     * @param border     the new border color
     */
    public synchronized void setBorder(Color border) {
        this.border = border;
    }

    /**
     * Returns the top margin.
     *
     * @return          the top margin
     */
    public int getTopMargin() {
        return topMargin;
    }

    /**
     * Returns the left margin.
     *
     * @return          the left margin
     */
    public int getLeftMargin() {
        return leftMargin;
    }

    /**
     * Returns the bottom margin.
```

```
     *
     * @return          the bottom margin
     */
    public int getBottomMargin() {
        return bottomMargin;
    }

    /**
     * Returns the right margin.
     *
     * @return          the right margin
     */
    public int getRightMargin() {
        return rightMargin;
    }

    /**
     * Used internally to calculate the minimum/maximum margin values.
     */
    private void resetMinMaxMargin() {
        maxMargin = topMargin;
        maxMargin = Math.max( maxMargin, leftMargin );
        maxMargin = Math.max( maxMargin, bottomMargin );
        maxMargin = Math.max( maxMargin, rightMargin );
        minMargin = topMargin;
        minMargin = Math.min( minMargin, leftMargin );
        minMargin = Math.min( minMargin, bottomMargin );
        minMargin = Math.min( minMargin, rightMargin );
    }

    /**
     * Sets all margins (top, left, bottom, and right) to the same, given value.
     *
     * @param margin         the margin
     */
    public synchronized void setMargins( int margin ) {
        topMargin = margin;
        leftMargin = margin;
        bottomMargin = margin;
        rightMargin = margin;
        resetMinMaxMargin();
    }

    /**
     * Sets all margins (top, left, bottom, and right) to the given values.
     *
     * @param top           the top margin
     * @param left          the left margin
     * @param bottom        the bottom margin
     * @param right         the right margin
     */
    public synchronized void setMargins( int top, int left, int bottom, int right ) {
        topMargin = top;
        leftMargin = left;
        bottomMargin = bottom;
        rightMargin = right;
        resetMinMaxMargin();
    }

    /**
     * Sets the top margin
     *
     * @param top           the top margin
     */
    public synchronized void setTopMargin( int margin ) {
        topMargin = margin;
        resetMinMaxMargin();
    }

    /**
     * Sets the left margin
     *
     * @param left          the left margin
     */
    public synchronized void setLeftMargin( int margin ) {
        leftMargin = margin;
        resetMinMaxMargin();
    }

    /**
     * Sets the bottom margin
     *
     * @param bottom        the bottom margin
     */
    public synchronized void setBottomMargin( int margin ) {
        bottomMargin = margin;
        resetMinMaxMargin();
    }

    /**
     * Sets the right margin
     *
     * @param right         the right margin
     */
    public synchronized void setRightMargin( int margin ) {
        rightMargin = margin;
        resetMinMaxMargin();
    }

    /**
     * Returns the current setting for the border thickness
     *
     * @return    the border thickness
     */
    public int getBorderThickness() {
        return borderThickness;
    }

    /**
     * Sets the border thickness
     *
     * @param borderThickness         the border thickness
     */
    public synchronized void setBorderThickness( int borderThickness ) {
        this.borderThickness = borderThickness;
    }

    /**
     * returns the left, right, top, and bottom insets of the background of the
     * current style, taking into account thickness and margins.
     *
     * @return  an Insets object containing the inset values
     */
    public Insets getInsets() {
```

```
        int top = borderThickness + topMargin;
        int left = borderThickness + leftMargin;
        int bottom = borderThickness + bottomMargin;
        int right = borderThickness + rightMargin;
        return new Insets( top, left, bottom, right );
    }

    /**
     * returns the left, right, top, and bottom insets of the background of the
     * current style, taking into account thickness and margins.
     *
     * @param   g           the Graphics in which to paint
     * @param   background  the current background color (or null), used if available to
     *                      calculate the border color if that is null.
     * @param   x           the x location of the upper-left point of the border
     * @param   y           the y location of the upper-left point of the border
     * @param   width       the width of the rectangle in which to draw the border
     * @param   height      the height of the rectangle in which to draw the border
     */
    public void paint( Graphics g, Color background, int x, int y, int width, int height ) {
        if ( border == null ) {
            if ( ( type == LINE ) ||
                 ( type == ROUND_RECT ) ) {
                if ( background == Color.black ) {
                    border = Color.white;
                }
                else {
                    border = Color.black;
                }
            }
            else {
                if ( background == null ) {
                    border = Color.lightGray;
                }
                else {
                    border = background;
                }
            }
        }

        if ( border != null ) {
            g.setColor( border );

            Color brighter = null;
            Color darker = null;

            switch( type ) {

                case THREED_IN:
                case THREED_OUT:
                case ETCHED_IN:
                case EMBOSSED_OUT: {
                    brighter = brighter(border);
                    darker = darker(border);
                    break;
                }
            }

            for ( int idx = 0; idx < borderThickness; idx++ ) {
                if ( type == LINE ) {
                    g.drawRect( x + idx, y + idx,
                                ( ( width - 1 ) - ( idx * 2 ) ),
                                ( ( height - 1 ) - ( idx * 2 ) ) );
                }
                else if ( type == ROUND_RECT ) {

                    int arcSize = ( minMargin * 8 - ( idx * 2 ) );
                    int rrx = x + idx;
                    int rry = y + idx;
                    int rrw = ( width - 1 ) - ( idx * 2 );
                    int rrh = ( height - 1 ) - ( idx * 2 );

                    g.drawRoundRect( x + idx, y + idx, rrw, rrh, arcSize, arcSize );
                    if ( ( idx + 1 ) < borderThickness ) {
                        g.drawRoundRect( rrx, rry, rrw, rrh - 1, arcSize - 1, arcSize );
                        g.drawRoundRect( rrx + 1, rry, rrw, rrh - 1, arcSize - 1, arcSize );
                        g.drawRoundRect( rrx, rry, rrw - 1, rrh, arcSize, arcSize - 1 );
                        g.drawRoundRect( rrx, rry + 1, rrw - 1, rrh, arcSize, arcSize - 1 );
                    }
                }
                else {
                    Color top = brighter;
                    Color bottom = darker;

                    if ( ( type == THREED_IN ) ||
                         ( type == ETCHED_IN ) ) {
                        top = darker;
                        bottom = brighter;
                    }

                    if ( ( type == ETCHED_IN ) ||
                         ( type == EMBOSSED_OUT ) ) {
                        if ( idx >= ( borderThickness / 2 ) ) {
                            Color temp = top;
                            top = bottom;
                            bottom = temp;
                        }

                    }

                    if ( ( idx == ( borderThickness - 1 ) ) &&
                         ( type == THREED_IN ) ) {
                        g.setColor( darker(top) );
                    }
                    else {
                        g.setColor( top );
                    }

                    g.drawLine( x + idx, y + idx,
                                x + idx,
                                y + ( ( height - 1 ) - ( idx ) ) );
                    g.drawLine( x + idx, y + idx,
                                x + ( ( width - 1 ) - ( idx ) ),
                                y + idx );

                    if ( ( ( idx == ( borderThickness - 1 ) ) &&
                           ( type == THREED_IN ) ) ||
                         ( ( idx == 0 ) &&
                           ( type == THREED_OUT ) ) ) {
                        g.setColor( darker(bottom) );
                    }
                    else {
                        g.setColor( bottom );
                    }
```

```
                    g.drawLine( x + idx,
                                y + ( ( height - 1 ) - ( idx ) ),
                                x + ( ( width - 1 ) - ( idx ) ),
                                y + ( ( height - 1 ) - ( idx ) ) );
                    g.drawLine( x + ( ( width - 1 ) - ( idx ) ),
                                y + idx,
                                x + ( ( width - 1 ) - ( idx ) ),
                                y + ( ( height - 1 ) - ( idx ) ) );
                }
            }
        }
    }
}
```

```
// ImageButton.java
// Third-party library class for a button with an image instead of text.

// Note that I have made changes to the code in a few places.
// Plain subclasses would not have worked well without an overhaul
// because the author used "private" methods throughout.


/**
 * Copyright(c) 1997 DTAI, Incorporated (http://www.dtai.com)
 *
 *                        All rights reserved
 *
 * Permission to use, copy, modify and distribute this material for
 * any purpose and without fee is hereby granted, provided that the
 * above copyright notice and this permission notice appear in all
 * copies, and that the name of DTAI, Incorporated not be used in
 * advertising or publicity pertaining to this material without the
 * specific, prior written permission of an authorized representative of
 * DTAI, Incorporated.
 *
 * DTAI, INCORPORATED MAKES NO REPRESENTATIONS AND EXTENDS NO WARRANTIES,
 * EXPRESS OR IMPLIED, WITH RESPECT TO THE SOFTWARE, INCLUDING, BUT
 * NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS FOR ANY PARTICULAR PURPOSE, AND THE WARRANTY AGAINST
 * INFRINGEMENT OF PATENTS OR OTHER INTELLECTUAL PROPERTY RIGHTS.  THE
 * SOFTWARE IS PROVIDED "AS IS", AND IN NO EVENT SHALL DTAI, INCORPORATED OR
 * ANY OF ITS AFFILIATES BE LIABLE FOR ANY DAMAGES, INCLUDING ANY
 * LOST PROFITS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES RELATING
 * TO THE SOFTWARE.
 */

import java.awt.*;
import java.awt.image.*;

/**
 * ImageButton - A button component with an image in it
 *
 * @author  DTAI, Incorporated
 */

public class ImageButton extends Canvas {

    public static final int UNARMED = 0;
    public static final int ARMED = 1;
    public static final int OVER = 2;
    public static final int DISABLED = 3;

    private static final Border defaultUnarmedBorder =
        new DefaultImageButtonBorder( false );
    private static final Border defaultArmedBorder =
        new DefaultImageButtonBorder( true );

    private MediaTracker tracker;

    private Image images[] = new Image[4];
    private Border borders[] = new Border[4];

    private boolean generatedDisabled = false;
    private boolean mousedown = false;

    private int buttonState = UNARMED;
```

```
    // added 03/12/97 by SHOE
    public Point                where = new Point(0,0);

    /**
     * Constructs an ImageButton
     */
    public ImageButton() {
        tracker = new MediaTracker( this );
        setUnarmedBorder( defaultUnarmedBorder );
        setArmedBorder( defaultArmedBorder );
    }

    /**
     * Constructs an ImageButton with the given image.
     *
     * @param image         the image for all states of the button
     *                      (until other images are assigned)
     */
    public ImageButton( Image image ) {
        this();
        setUnarmedImage( image );
    }

    /**
     * Used internally to add the Image to the array and the MediaTracker,
     * start loading the image if necessary via the tracker's "checkID", and
     * repaint if necessary.
     *
     * @param id        the buttonState id (also used as image id for the MediaTracker)
     * @param image     the image, which is not supposed to be null
     */
    private synchronized void setImage( int id, Image image ) {
        if ( images[id] != image ) {
            images[id] = image;
            if ( image != null ) {
                tracker.addImage( image, id );
                tracker.checkID( id, true );
            }
            if ( buttonState == id ) {
                repaint();
            }
        }
    }

    /**
     * Sets the image to display when the button is not pressed or hilited
     * because of a mouse-over.  This image is also used in those other cases
     * if no alternative image is requested.
     *
     * @param image     the unarmed image
     */
    public void setUnarmedImage( Image image ) {
        setImage( UNARMED, image );
        if ( images[ARMED] == null ) {
            setArmedImage( image );
        }
        if ( images[OVER] == null ) {
            setOverImage( image );
        }
        if ( ( images[DISABLED] == null ) ||
             generatedDisabled ) {
```

```
            setDisabledImage( null );
        }
    }

    /**
     * Sets the image to display when the button is pressed and the mouse
     * is still over the button.
     *
     * @param image     the armed image
     */
    public void setArmedImage( Image image ) {
        if ( image != null ) {
            setImage( ARMED, image );
        }
        else {
            setImage( ARMED, images[UNARMED] );
        }
    }

    /**
     * Sets the image to display when the button is not pressed and the mouse
     * is over the button.
     *
     * @param image     the over image
     */
    public void setOverImage( Image image ) {
        if ( image != null ) {
            setImage( OVER, image );
        }
        else {
            setImage( OVER, images[UNARMED] );
        }
    }

    /**
     * Sets the image to display when the button is disabled.
     *
     * @param image     the disabled image
     */
    public void setDisabledImage( Image image ) {
        generatedDisabled = false;
        if ( ( image == null ) &&
             ( images[UNARMED] != null ) ) {
            generatedDisabled = true;
            image = createImage( new FilteredImageSource( images[UNARMED].getSource(),
                                                          new DisableImageFilter() ) );
        }
        setImage( DISABLED, image );
    }

    /**
     * Gets the image to display when the button is not pressed or hilited
     * because of a mouse-over.  This image is also used in those other cases
     * if no alternative image is requested.
     *
     * @return     the unarmed image
     */
    public Image getUnarmedImage() {
        return ( images[UNARMED] );
    }

    /**
```

```
     * Gets the image to display when the button is pressed and the mouse
     * is still over the button.
     *
     * @return     the armed image
     */
    public Image getArmedImage() {
        return ( images[ARMED] );
    }

    /**
     * Gets the image to display when the button is not pressed and the mouse
     * is over the button.
     *
     * @return     the over image
     */
    public Image getOverImage() {
        return ( images[OVER] );
    }

    /**
     * Gets the image to display when the button is disabled.
     *
     * @return     the armed image
     */
    public Image getDisabledImage() {
        return ( images[DISABLED] );
    }

    /**
     * Used internally to add the Border to the array and repaint if necessary.
     *
     * @param   id      the buttonState, used to index the array
     * @param   border  the Border, which is not supposed to be null
     */
    private synchronized void setBorder( int id, Border border ) {
        if ( borders[id] != border ) {
            borders[id] = border;
            if ( buttonState == id ) {
                repaint();
            }
        }
    }

    /**
     * Sets the border to display when the button is not pressed or hilited
     * because of a mouse-over.  This border is also used in those other cases
     * if no alternative border is requested.
     *
     * @param border     the unarmed border
     */
    public void setUnarmedBorder( Border border ) {
        setBorder( UNARMED, border );
        if ( borders[ARMED] == null ) {
            setArmedBorder( border );
        }
        if ( borders[OVER] == null ) {
            setOverBorder( border );
        }
        if ( borders[DISABLED] == null ) {
            setDisabledBorder( border );
        }
    }
    /**
     * Sets the border to display when the button is pressed and the mouse
     * is still over the button.
     *
     * @param border     the armed border
     */
    public void setArmedBorder( Border border ) {
        if ( border != null ) {
            setBorder( ARMED, border );
        }
        else {
            setBorder( ARMED, borders[UNARMED] );
        }
    }

    /**
     * Sets the border to display when the button is not pressed and the mouse
     * is over the button.
     *
     * @param border     the over border
     */
    public void setOverBorder( Border border ) {
        if ( border != null ) {
            setBorder( OVER, border );
        }
        else {
            setBorder( OVER, borders[UNARMED] );
        }
        setBorder( OVER, border );
    }

    /**
     * Sets the border to display when the button is disabled.
     *
     * @param border     the disabled border
     */
    public void setDisabledBorder( Border border ) {
        if ( border != null ) {
            setBorder( DISABLED, border );
        }
        else {
            setBorder( DISABLED, borders[UNARMED] );
        }
        if ( buttonState == DISABLED ) {
            repaint();
        }
    }

    /**
     * Gets the border to display when the button is not pressed or hilited
     * because of a mouse-over.  This border is also used in those other cases
     * if no alternative border is requested.
     *
     * @return     the unarmed border
     */
    public Border getUnarmedBorder() {
        return ( borders[UNARMED] );
    }

    /**
     * Gets the border to display when the button is pressed and the mouse
```

```
     * is still over the button.
     *
     * @return     the armed border
     */
    public Border getArmedBorder() {
        return ( borders[ARMED] );
    }

    /**
     * Gets the border to display when the button is not pressed and the mouse
     * is over the button.
     *
     * @return     the over border
     */
    public Border getOverBorder() {
        return ( borders[OVER] );
    }

    /**
     * Gets the border to display when the button is disabled.
     *
     * @return     the armed border
     */
    public Border getDisabledBorder() {
        return ( borders[DISABLED] );
    }

    /**
     * Gets the current buttonState id for the button
     *
     * @return     the button state integer id
     */
    public int getButtonState() {
        return buttonState;
    }

    /**
     * Sets the current buttonState id for the button
     *
     * @param   buttonState     the button state integer id
     */
    protected void setButtonState( int buttonState ) {
        if ( buttonState != this.buttonState ) {
            this.buttonState = buttonState;
            repaint();
        }
    }

    /**
     * Overrides awt.Component.disable() to also set the button state.
     */
    public void disable() {
        setButtonState( DISABLED );
        super.disable();
    }

    /**
     * Overrides awt.Component.enable() to also set the button state.
     */
    public void enable() {
        setButtonState( UNARMED );
        super.enable();
    }

    /**
     * Overrides awt.Component.paint() to paint the current border and image.
     *
     * @param     g     The Graphics in which to draw
     */
    public void paint( Graphics g ) {
        Dimension size = size();
        borders[buttonState].paint( g, getBackground(), 0, 0, size.width, size.height );
        try {
            if ( ! tracker.checkID( buttonState ) ) {
                tracker.waitForID( buttonState );
            }
            if ( ! tracker.isErrorID( buttonState ) ) {
                Insets insets = borders[buttonState].getInsets();
                int imageWidth = images[buttonState].getWidth( this );
                int imageHeight = images[buttonState].getHeight( this );
                int x = insets.left +
                        ( ( ( size.width - ( insets.left + insets.right ) ) -
                              imageWidth ) / 2 );
                int y = insets.top +
                        ( ( ( size.height - ( insets.top + insets.bottom ) ) -
                              imageHeight ) / 2 );
                g.drawImage( images[buttonState], x, y, this );
            }
        }
        catch ( InterruptedException ie ) {
        }
    }


    /**
     * Overrides awt.Component.preferredSize() to return the preferred size of the button.
     * This assumes the images (if more than one) are all the same size.  It also calculates
     * the maximum insets from all borders and adds them to the image dimensions.
     *
     * @param     g     The Graphics in which to draw
     */
    public Dimension preferredSize() {
        Dimension pref = new Dimension();
        try {
            if ( ! tracker.checkID( buttonState ) ) {
                tracker.waitForID( buttonState );
            }
            if ( ! tracker.isErrorID( buttonState ) ) {
                Dimension size = size();
                pref.width = images[buttonState].getWidth( this );
                pref.height = images[buttonState].getHeight( this );
            }
            int maxWidthAdd = 0;
            int maxHeightAdd = 0;
            for ( int i = 0; i < DISABLED; i++ ) {
                Insets insets = borders[i].getInsets();
                maxWidthAdd = Math.max( maxWidthAdd, insets.left+insets.right );
                maxHeightAdd = Math.max( maxHeightAdd, insets.top+insets.bottom );
            }
            pref.width += maxWidthAdd;
            pref.height += maxHeightAdd;
        }
        catch ( InterruptedException ie ) {
        }
```

```
        return pref;
    }

    /**
     * Overrides awt.Component.mouseDown() to arm the button.
     *
     * @param   evt     The mouse down event
     * @param   x       The mouse x position
     * @param   y       The mouse y position
     * @return  true if the event was handled
     */
    public boolean mouseDown( Event evt, int x, int y ) {
        mousedown = true;
        setButtonState( ARMED );
        return true;
    }

    /**
     * Overrides awt.Component.mouseExit()  to disarm the button if the mouse is
     * down, or unhilite it if a special OVER image and/or border was set.
     *
     * @param   evt     The mouse exit event
     * @param   x       The mouse x position
     * @param   y       The mouse y position
     * @return  true if the event was handled
     */
    public boolean mouseExit( Event evt, int x, int y ) {
        setButtonState( UNARMED );
        return true;
    }

    /**
     * Overrides awt.Component.mouseEnter() to rearm the button if the mouse is
     * down, or hilite it if a special OVER image and/or border was set.
     *
     * @param   evt     The mouse enter event
     * @param   x       The mouse x position
     * @param   y       The mouse y position
     * @return  true if the event was handled
     */
    public boolean mouseEnter( Event evt, int x, int y ) {
        if ( mousedown ) {
            setButtonState( ARMED );
        }
        else {
            setButtonState( OVER );
        }
        return true;
    }

    /**
     * Overrides awt.Component.mouseUp() and invokes "action" if the button was ARMED
     * (because the mouse was over the button when it was released).
     *
     * @param   evt     The mouse up event
     * @param   x       The mouse x position
     * @param   y       The mouse y position
     * @return  true if the event was handled
     */
    public boolean mouseUp( Event evt, int x, int y ) {
        mousedown = false;
        if ( inside( x, y ) ) {
```

```
        setButtonState( OVER );
        if ( ! action( evt, evt.arg ) ) {
            Container parent = getParent();
            while ( ( parent != null ) &&
                    ( ! parent.action( evt, evt.arg ) ) ) {
                parent = parent.getParent();
            }
        }
    }
    return true;
}

/   dded by SHOE, 03/08/97
p   ic void CleanUpImages()
{
    int         index, index2;
    boolean     flushedBefore;

    for(index = 0; index < 4; index++)
    {
        if(images[index] != null) {
            flushedBefore = false;
            for(index2 = 0; index2 < index; index2++)
            {
                if(images[index] == images[index2])
                    flushedBefore = true;
            }
            if(!flushedBefore)
                images[index].flush();
        }
    }
    for(index = 0; index < 4; index++)
    {
        images[index] = null;
        borders[index] = null;
    }
}

/   dded by SHOE, 03/13/97
p   ic Point GetDesiredPosition()
{
    return where;
}

/   dded by SHOE, 03/21/97
p   ic void layout()
{
    ReshapeButton();
}

/   dded by SHOE, 03/21/97
p   ic void SetPosition(int inX, int inY)
{
    where.x = inX;
    where.y = inY;
}

/   dded by SHOE, 03/21/97
p   ic void SetPosition(Point coords)
{
    if(coords != null)
        where = coords;
```

```
    }

    // added by SHOE, 03/21/97
    public void MoveButton()
    {
        move(where.x, where.y);
    }

    // added by SHOE, 03/21/97
    public void ReshapeButton()
    {
        Dimension   size = preferredSize();
        reshape(where.x, where.y, size.width, size.height);
    }
}

/**
 * DisableImageFilter - an RGBImageFilter that "greys out" an image by "blanking out"
 * every other pixel.
 */
class DisableImageFilter extends RGBImageFilter
{
    /**
     * Constructs a DisableImageFilter.  The canFilterIndexColorModel is set to false
     * because the pixel index is important during filtering.
     */
    public DisableImageFilter() {
        canFilterIndexColorModel = false;
    }

    /**
     * Called when a disabled image is created to alter the pixels to be blanked out.
     *
     * @param   x   the x position of the pixel
     * @param   y   the y position of the pixel
     * @param   rgb the rgb value of the pixel
     */
    public int filterRGB( int x, int y, int rgb ) {
        if ( ( ( x % 2 ) ^ ( y % 2 ) ) == 1 ) {
            return ( rgb & 0xffffff );
        }
        else {
            return rgb;
        }
    }
}

/**
 * DefaultImageButtonBorder - a Border, subclassed to set the default border values.
 */
class DefaultImageButtonBorder extends Border {

    public DefaultImageButtonBorder( boolean armed ) {
        setBorderThickness( 2 );
        if ( armed ) {
            setType( THREED_IN );
            setMargins( 4, 4, 2, 2 );
        }
        else {
            setType( THREED_OUT );
            setMargins( 3 );
        }
	}
}
```

```
// ScreenButton.java
// This subclasses ImageButton for special use on screens only.

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

public class ScreenButton extends ImageButton {
    private Screen      parentScreen = null;
    boolean             selected = false;   // never actually used

    public static final int     NONE = 0;   // btn type, still unused
    public static final int     SURROUND = 1;
    public static final int     LIGHT = 2;
    int                         selType = NONE;


    // None of the methods below really require much explanation.

    public ScreenButton() {
        super();
    }

    public ScreenButton(Screen s)
    {
        super();
        SetParentScreen(s);
    }

    public ScreenButton( Image image ) {
        super(image);
    }

    public ScreenButton(Screen s, Image image)
    {
        super(image);
        SetParentScreen(s);
    }

    public ScreenButton(Image image, int inX, int inY)
    {
        super(image);
        SetPosition(inX, inY);
    }

    public ScreenButton(Screen s, Image image, int inX, int inY)
    {
        super(image);
        SetPosition(inX, inY);
        SetParentScreen(s);
    }

    public void SetSelectionType(int type)
    {
        if((type >= 0) && (type < 3))
        {
            selType = type;
        }
    }
```

```
    public int GetSelectionType()
    {
        return selType;
    }

    public void SelectButton()
    {
        selected = true;
        getParent().repaint();
    }

    public void DeselectButton()
    {
        selected = false;
        getParent().repaint();
    }

    public boolean IsSelected()
    {
        return selected;
    }

    public void update(Graphics g)
    {
        if(parentScreen != null)
        {
            AppFrame    f = parentScreen.GetScreenFrame();
            if(f != null)
            {
                if(f.offImage != null)
                {
                    Graphics offG = f.offImage.getGraphics();

                    paint(offG);
                    g.drawImage(f.offImage, 0, 0, this);
                }
            }
        }
    }

    public void SetParentScreen(Screen s)
    {
        parentScreen = s;
    }

    public Screen GetParentScreen()
    {
        return parentScreen;
    }
}
```

References

Alba et al. (1996), "Interactive Home Shopping and the Retail Industry, " Report for the Marketing Science Institute, (July), 1.

Benabadji, Ahmed (1997), "Building Trust in the Electronic Marketplace," Master of Business Administration thesis, MIT Sloan School of Management, Massachusetts Institute of Technology.

Boone, Barry (1996), *Java Essentials for C and C++ Programmers*, Addison-Wesley Developers Press, Reading, MA.

Boone, Barry and Dave Mark (1996), *Learn Java on the Macintosh*, Addison-Wesley Developers Press, Reading, MA.

Butler, J. K., and R. S. Cantrell (1984), "A Behavioral Decision Theory Approach to Modeling Dyadic Trust in Superiors and Subordinates," *Psychological Reports*, 55.

Deutsch, M. (1958), "Trust and Suspicion," *Journal of Conflict Resolution*.

Hosmer, L. T. (1995), "Trust: The Connecting Link between Organizational Theory and Philosophical Ethics," *Academy of Management Review*, Vol. 20, No. 2.

References

Hsu, James M. (1996), "Programming Surveys for the MIT Information Acceleration Project," Master of Engineering thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Newman, Alexander, et al. (1996), *Special Edition Using Java*, Que Corporation, Indianapolis, IN.

Sun Microsystems, JavaSoft Home Page, http://www.javasoft.com.

Urban, Glen L., Bruce D. Weinberg, and John R. Hauser (1996), "Premarket Forecasting of Really-New Products," *Journal of Marketing*, 60 (January), 47-60.

Van Slyke, C. (1996), "Trust Between Users and Their Intelligent Agents," Working paper, University of South Florida.

Withers, John (1997), *Developing Java Entertainment Applets*, John Wiley & Sons, Inc., New York, NY.