and 14. For now, it's worth noting various key issues that impact on the decisions made about the technical components.

Any decision made about how one component might map into the technical architecture will affect the decisions about other components. This can occur for a number of reasons, but it is often because a technical component provides broad-based support or puts constraints on other components. For example, the WebE team may choose to design **SafeHomeAssured.com** in a way that stores product information as XML files. Later, the team discovers that the content management system doesn't easily support access to XML content, but rather assumes that the content will be stored in a conventional relational database. One component of the technical architecture conflicts with constraints imposed by another component.

Many (possibly most) technical components are developed without a single narrow focus in mind. Rather, they are designed to be as broadly applicable as possible. Hence these components will often provide considerable flexibility, but also significant complexity. A fully blown commercial-strength content management system would obviously provide the functionality required for a simple static website, but would also provide much additional functionality that is not needed. This unnecessary functionality can then complicate the development—distracting the developers and requiring additional effort to manage. In some cases, a much simpler technical solution will be more appropriate.

Finally, the specific choices of technical components that make up the technical architecture depend not just on the specific needs imposed by the conceptual architecture, but also on the environment in which it will be operating (e.g., the WebApp hosting environment might be based on a Linux platform) and the skills of the development team.

Where We've Been . . . Where We're Going

WebApp design is a combination of art and engineering. To be effective, a good design must accommodate each of the requirements established by stakeholders during the communication activity. But it must also accommodate the inevitable changes to requirements that will occur throughout the design and into construction and delivery. Design begins by focusing on the way in which a user will interact with an application and then moves to consider the functions and information content that will be required to meet stakeholders' needs. It then proceeds to physical design, in which the logical elements are mapped into a representation that can be implemented as part of the WebApp. The primary output of the design activity is a design model that encompasses interface descriptions, aesthetics, content, navigation, architecture, and component-level design issues.

Every WebApp designer must be cognizant of a set of generic design goals: simplicity, consistency, identity, robustness, navigability, visual appeal, and compatibility. In addition, a variety of quality frameworks can be applied to WebApp design. These quality criteria provide a set of objectives for achieving WebApp design quality. In addition, a user-centric quality model provides a solid indication of the degree to which end users like the technology that the WebApp delivers. WebApp designers can apply a Design IQ Checklist that serves to assess the importance of various quality criteria and the degree to which the WebE team has met each of the criteria.

The design process is an agile, iterative collection of design actions that are applied to each WebApp increment as it is created. A design pyramid can be used to describe a set of design actions that are performed (with varying degrees of emphasis) for each WebApp increment. The design actions include interface design, aesthetic design, content design, navigation design, functional design, architecture design, and component design.

Each of the design actions represents a challenge for the WebE team. In the chapters that follow, we'll consider the mechanics of each design action in some detail.

References

[Bec99] Beck, K., *Extreme Programming Explained: Embrace Change,* Addison-Wesley, 1999.

[Bec05] Beck, K., and C. Andres, *Extreme Programming Explained: Embrace Change,* 2nd ed., Addison-Wesley, 2005.

[Cro07] Cross, M., *Developer's Guide to Web Application Security,* Syngress, 2007.

[Dav89] Davis, F. D., "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly,* vol. 13, no. 3, 1989, pp. 319–342.

[Das07] Daswani, N., et al., *Foundations of Security,* Apress, 2007.

[Kai02] Kaiser, J., "Elements of Effective Web Design," About, Inc., 2002, **http://webdesign.about.com/library/weekly/aa091998.htm** (accessed September 15, 2004).

[Kal03] Kalman, S., *Web Security Field Guide,* Cisco Press, 2003.

[Mil00] Miller, E., "The Website Quality Challenge," *Software Research, Inc.,* 2000, **www.soft.com/eValid/Technology/White.Papers/website.quality.challenge.html** (accessed July 31, 2007).

[Nie00] Nielsen, J., *Designing Web Usability,* New Riders Publishing, 2000.

[Off02] Offutt, J., "Quality Attributes of Web Software Applications," *IEEE Software,* March/April, 2002, pp. 25–32.

[Ols99] Olsina, L. et al., "Specifying Quality Characteristics and Attributes for Web Sites," *Proc. 1st ICSE Workshop on Web Engineering*, ACM, Los Angeles, May 1999.

[Pow00] Powell, T., *Web Design*, McGraw-Hill/Osborne, 2000.

[Til00] Tillman, H. N., "Evaluating Quality on the Net," *Babson College*, May 30, 2000, **www.hopetillman.com/findqual.html#2.**

[Ven00] Venkatesh, V., and F. D. Davis, "Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies," *Management Science* vol. 46, no. 2, February 2000, pp. 186–204.

[Ven03] Venkatesh, V., M. G. Morris, G. B. Davis, and F. D. Davis, "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly*, vol. 27, no. 3, 2003, pp. 425–478.

CHAPTER 9

INTERACTION DESIGN

Every user interface—whether it is designed for a WebApp, a traditional software application, a consumer product, or an industrial device—should exhibit the following characteristics: easy to use, easy to learn, easy to navigate, intuitive, consistent, efficient, error-free, and functional. It should provide the end user with a satisfying and rewarding experience. Interface design concepts, principles, and methods provide a Web engineer with the tools required to achieve this list of attributes.

Interaction design for WebApps begins not with a consideration of technology or tools, but with a careful examination of the end user. During analysis modeling (Chapter 7), a user hierarchy was developed. Each user category may have subtly different needs, may want to interact with the WebApp in different ways, and may require unique functionality and content. This information is derived during the communication activity (Chapter 4), may be refined during analysis modeling, and is revisited as the first step in interaction design.

Dix [Dix99] argues that you should design an interface so that it answers three primary questions for the end user:

Where am I? The interface should (1) provide an indication of the WebApp that has been accessed[1], and (2) inform users of their location in the content hierarchy.

What can I do now? The interface should always help users understand their current options—what functions are available, what links are live, what content is relevant?

Where have I been, where am I going? The interface must facilitate navigation. Hence, it must provide a "map" (implemented in a way that is easy to understand) of where users have been and what paths they may take to move elsewhere within the WebApp.

An effective WebApp interface must provide answers for each of these questions as end users navigate through content and functionality.

1 Each of us has bookmarked a website page, only to revisit it later and have no indication of the website or the context for the page (as well as no way to move to another location within the site).

Interface Design Principles and Guidelines

The user interface of a WebApp is its "first impression." Regardless of the value of its content, the sophistication of its processing capabilities and services, and the overall benefit of the WebApp itself, a poorly designed interface will disappoint the potential user and may, in fact, cause the user to go elsewhere. Because of the sheer volume of competing WebApps in virtually every subject area, the interface must grab a potential user immediately.

Bruce Tognozzi [Tog01] defines a set of fundamental characteristics that all interfaces should exhibit and, in doing so, establishes a philosophy that should be followed by every WebApp interface designer:

> Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work.
>
> Effective interfaces do not concern the user with the inner workings of the system. Work is carefully and continuously saved, with full option for the user to undo any activity at any time.
>
> Effective applications and services perform a maximum of work, while requiring a minimum of information from users.

You might argue that everybody recognizes these things and every interface designer wants to achieve them. And yet, each of us has seen more than a few really bad interfaces and, surely, will continue to see many more.

What Principles Do We Apply to Design Effective Interfaces?

In order to design an effective interface, Tognozzi [Tog01] identifies a set of overriding design principles:[2]

Anticipation. *A WebApp should be designed so that it anticipates the user's next move.* For example, consider a customer support WebApp developed by a manufacturer of computer printers. A user has requested a content object that presents information about a printer driver for a newly released operating system. The designer of the WebApp should anticipate that the user might request a download of the driver and should provide navigation facilities that allow this to happen without requiring the user to search for this capability.

Communication. *The interface should communicate the status of any activity initiated by the user.* Communication can be obvious (e.g., a text message) or subtle (e.g., a sheet of paper moving through a printer to indicate that printing is under way). The interface should also communicate the user's status (e.g., the user's identification) and location within the WebApp content hierarchy.

Consistency. *The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout) should be consistent throughout the WebApp.* For example, if underlined blue text implies a navigation link, content should never incorporate blue underlined text that does *not* imply a link. In addition, an object, say a yellow triangle, used to indicate a caution message before the user invokes a particular function or action, should not be used for other purposes elsewhere in the WebApp. Finally, every feature of the interface should respond in a manner that is consistent with user expectations.[3]

Controlled autonomy. *The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application.* For example, navigation to secure portions of the WebApp should be controlled by a user ID and a password, and there should be no navigation mechanism that enables a user to circumvent these controls.

Efficiency. *The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the Web engineer who designs and builds it or the client-server environment that executes it.* Tognozzi [Tog01] discusses this when he writes:

> This simple truth is why it is so important for everyone involved in a software project to appreciate the importance of making user productivity goal one and to understand the vital difference between building an efficient system and empowering an efficient user. This truth is also key to the need for close and constant cooperation, communication, and conspiracy between engineers and human interface designers if this goal is to be achieved.

Flexibility. *The interface should be flexible enough to enable some users to accomplish tasks directly and others to explore the WebApp in a somewhat random fashion.* In every case, it should enable users to understand where they are and provide users with the functionality to undo mistakes and retrace poorly chosen navigation paths.

Focus. *The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand.* In all hypermedia there is a tendency to route the user to loosely related content. Why? Because it's very easy to do! The

2 Tognozzi's original principles have been adapted and extended for use in this book. See [Tog01] for further discussion of these principles.

3 Tognozzi [Tog01] notes that the only way to be sure that user expectations are properly understood is through comprehensive user testing (Chapter 16).

problem is that users can rapidly become lost in many layers of supporting information and lose sight of the original content that they wanted in the first place. The WebApp interface and its content should stay focused on whatever objective users must accomplish.

Fitt's law. *"The time to acquire a target is a function of the distance to and size of the target"* [Tog01]. Based on a study conducted in the 1950s [Fit54], Fitt's law "is an effective method of modeling rapid, aimed movements, where one appendage (like a hand) starts at rest at a specific start position, and moves to rest within a target area." If a sequence of selections or standardized inputs (with many different options within the sequence) is defined by a user task, the first selection (e.g., mouse pick) should be physically close to the next selection. For example, a WebApp home page interface for an e-commerce site that sells consumer electronics provides the following menu options:

Get a gift suggestion

Buy a product

Comparison shop

Ask a question

Get technical support

Each of these options implies a set of follow-on user choices or actions. For example, the "Buy a product" option requires that the user enter a product category followed by the product name. The product category (e.g., audio equipment, televisions, DVD players) appears as a pull-down menu as soon as "Buy a product" is selected. The next choice is then immediately obvious (it is nearby), and the time to acquire it is negligible. If, on the other hand, the choice appeared on a menu that was located on the other side of the screen, the time for the user to acquire it (and then make the choice) would be far too long.

User interface objects. *A vast library of reusable human interface objects (and patterns) has been developed for WebApps. Use them.* Any interface object that can be "seen, heard, touched or otherwise perceived" [Tog01] by an end user can be acquired from any one of a number of object libraries.

Latency reduction. *Rather than making the user wait for some internal operation to complete (e.g., downloading a complex graphical image), the WebApp should use multitasking in a way that lets the user proceed with work as if the operation has been completed.* In addition to reducing latency, delays must be acknowledged so that the user understands what is happening. This

includes (1) providing audio feedback (e.g., a click or bell tone) when a selection does not result in an immediate action by the WebApp, (2) displaying an animated clock or progress bar to indicate that processing is under way, and (3) providing some entertainment (e.g., an animation or text presentation) while lengthy processing occurs.

Learnability. *A WebApp interface should be designed to minimize learning time and, once learned, to minimize relearning required when the WebApp is revisited.* In general the interface should emphasize a simple, intuitive design that organizes content and functionality into categories that are obvious to the user.

Metaphors. *An interface that uses an interaction metaphor is easier to learn and easier to use, as long as the metaphor is appropriate for the application and the user.* A metaphor should call on images and concepts from the user's experience, but it does not need to be an exact reproduction of a real-world experience. For example, an e-commerce site that implements automated bill paying for a financial institution uses a checkbook metaphor (not surprisingly) to assist the user in specifying and scheduling bill payments. However, when "writing" a check, the user need not enter the complete payee name but can pick from a list of payees or have the system select a payee based on the first few typed letters. The metaphor remains intact, but the user gets an assist from the WebApp.

Maintain work product integrity. *A work product (e.g., a form completed by the user, a user-specified list) must be automatically saved so that it will not be lost if an error occurs.* Each of us has experienced the frustration associated with completing a lengthy WebApp form only to have the content lost because of an error (made by us, by the WebApp, or in transmission from client to server). To avoid this, a WebApp should be designed to auto-save all user-specified data. The interface should support this function and provide the user with an easy mechanism for recovering "lost" information.

Readability. *All information presented through the interface should be readable by young and old.* The interface designer should emphasize readable type styles, font sizes, and color background choices that enhance contrast.

Track state. *When appropriate, the state of user interactions should be tracked and stored so that users can log off and return later to pick up where they left off.* In general, cookies can be designed to store state information. However, cookies are a controversial technology, and other design solutions may be more palatable for some users.

Visible navigation. *A well-designed WebApp interface provides "the illusion that users are in the same place, with the work brought to them"* [Tog01]. When

this approach is used, navigation is not a user concern. Rather, the user retrieves content objects and selects functions that are displayed and executed through the interface.

Each of these design principles is applied as the preliminary interface design is created and when the design is reviewed (see sidebar).

SAFEHOME

Interface Team Walkthrough

The scene: SafeHomeAssured.com team leader's office

The players: Team leader and two members of the WebE team

The conversation:

Team member 1 (to team leader): Have you had a chance to take a look at our preliminary interface design [Figure 7.9] for **SafeHomeAssured.com?**

Team leader: Yeah . . . I went through it from a technical point of view, and I have a bunch of notes. I e-mailed 'em to you this morning.

Team member 1: Haven't looked at my e-mail yet . . . give me a summary of the important issues.

Team leader: Overall, you've done a good job, nothing groundbreaking, but it's a typical interface, decent aesthetics, reasonable layout. You hit all the important functions . . .

Team member 2 (smiling ruefully): But?

Team leader: Well, there are a few things . . . .

Team member 1: Such as . . .

Team leader: You guys remember the very first iteration on the home page where the major functions displayed in the menu were

About the company
Describe your home
Get *SafeHome* component recommendations
Purchase a product
Get technical support

The problem was that we left out some things, and those functions weren't at the same level of abstraction. So we retooled the whole thing, and you come up with this [points at the preliminary prototype from Figure 7.9].

Team member 2 (smiling): I like it.

Team leader: So do I, but marketing wants greater emphasis on the company and wants distinct photos of the product and services. So . . . I think our main menu should be:

About the company
Home security
Monitoring services
Our technology
Contact us

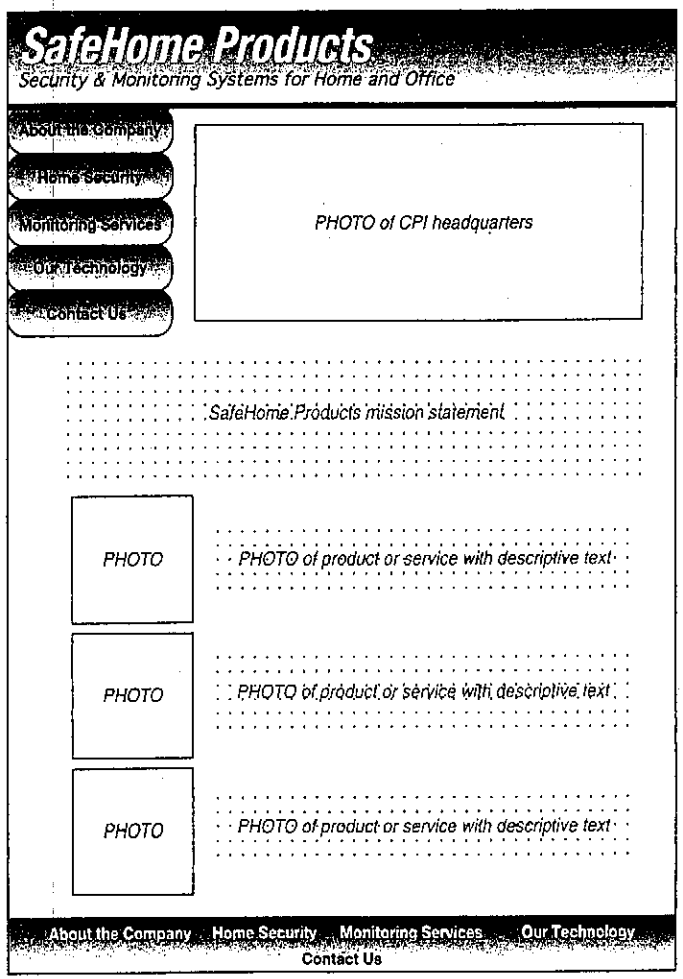
There'll be lots of subfunctions under each of these major options. Might think about having them pop us as a submenu when the top-level option is chosen. Each of the submenu options will lead the user to a separate Web page with its own layout, content objects, and functions.

Team member 1: But don't we want to register users who want to go directly to pages that meet their needs? We have no direct mechanism to do that here.

Team leader: You're right. It might not be a bad idea to provide a log-in like we had on the previous interface prototype iteration. I'll run it past marketing, but I think we can proceed as is for now.

FIGURE 9.1

Home page preliminary design layout.



As a result of modifications suggested during the preliminary interface design review, the WebE team produces a new home page interface design,[4] as illustrated in Figure 9.1.

4 Recall that we noted that the preliminary interface design (Figure 7.9) would likely change as the WebE team moved further into design.

What About Some Pragmatic Design Guidelines?

Nielsen and Wagner [Nie96] suggest a few pragmatic interface design guidelines (based on their redesign of a major WebApp) that provide a nice complement to the principles suggested earlier:

- Reading speed on a computer monitor is approximately 25 percent slower than reading speed for hard copy. Therefore, *do not force the user to read voluminous amounts of text,* particularly when the text explains the operation of the WebApp or assists in navigation.
- *Avoid "under construction" signs*—they raise expectations and cause an unnecessary link that is sure to disappoint or frustrate users.
- *Users prefer not to scroll.* Important information should be placed within the dimensions of a typical browser window.
- *Navigation menus and head bars should be designed consistently and should be available on all pages that are available to the user.* The design should not rely on browser functions (e.g., the back arrow) to assist in navigation.
- *Aesthetics should never supersede functionality.* For example, a simple button might be a better navigation option than an aesthetically pleasing, but vague image or icon whose intent is unclear.
- *Navigation options should be obvious, even to the casual user.* The user shouldn't have to search the screen to determine how to link to other content or services.

A well-designed interface the improves the user's perception of the content or services provided by the site. It need not necessarily be flashy, but it should always be well structured and ergonomically sound.

Interface Design Workflow

Interface design for WebApps begins with the identification of user, task, and environmental requirements. These activities are performed as part of requirements gathering and modeling discussed earlier in this book. Once user tasks have been identified, user scenarios (use cases) are created and analyzed to define a set of interface objects and actions.

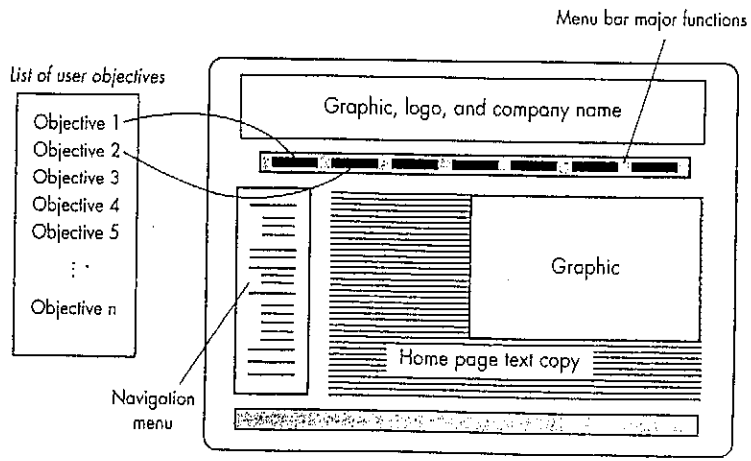
Information contained within the analysis model (Chapter 7) forms the basis for the creation of a screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Tools are then used to prototype and ultimately implement the interface design model.

The following tasks represent a rudimentary workflow for WebApp interface design:

1. **Review user characteristics and categories, user tasks, use cases, and related information contained in the analysis model and refine as required.**
2. **Develop a rough design prototype of the WebApp interface layout.** An interface prototype (including the layout) may have been developed as part of the analysis modeling activity. If the layout already exists, it should be reviewed and refined as required. If the interface layout has not been developed, the WebE team should work with stakeholders to develop it at this time. A preliminary interface design for **SafeHomeAssured.com** (a modified version of the prototype created as part of analysis modeling) is shown in Figure 9.1.
3. **Map user objectives into specific interface actions.** For the vast majority of WebApps, the user will have a relatively small set of primary objectives (typically, between four and seven primary objectives). These should be mapped into specific interface actions as shown in Figure 9.2. In essence, the design must answer the following question: "How does the interface enable the user to accomplish each objective or provide the launching point for relevant tasks?"
4. **Define a set of user tasks that are associated with each action.** Each interface action (e.g., "Buy a product") is associated with a set of user tasks. These tasks have been identified during analysis. During design, they must

FIGURE 9.2

Mapping user objectives into interface actions.



be mapped into specific interactions that encompass navigation issues, content objects, and WebApp functions.

5. **Develop screen images for each interface action.** As each action is considered, a sequence of screen images should be created to depict how the interface responds to user interaction. Content objects should be identified (even if they have not yet been designed and developed), WebApp functionality should be shown, and navigation links should be indicated.
6. **Refine interface layout and screen images using input from aesthetic design.** Web engineers complete a rough layout, but the aesthetic look and feel for a major commercial site is often developed by artistic, rather than technical, professionals—such as graphic designers. Aesthetic design (discussed later in this chapter) is integrated with the work performed by the interface designer.
7. **Identify user interface objects that are required to implement the interface.** This task may require a search through an existing class library to find those reusable objects (classes) that are appropriate for the WebApp interface. In addition, any custom classes are specified at this time.
8. **Develop a procedural representation of the user's interaction with the interface.** This optional task uses UML sequence diagrams and/or activity diagrams to depict the flow of activities (and decisions) that occurs as the user interacts with the WebApp.
9. **Develop a behavioral representation of the interface.** This optional task makes use of UML state diagrams to represent state transitions and the events that cause them. Control mechanisms (i.e., the objects and actions available to the user to alter a WebApp state) are defined.
10. **Describe the interface layout for each state.** Using design information developed in tasks 2 and 5, associate a specific layout or screen image with each WebApp state described in task 9.

Pair walkthroughs (Chapter 5) should be conducted throughout all these design tasks and should focus on usability. In addition, it's important to note that the design workflow chosen by a WebE team should be adapted to the special requirements of the application that is to be built. Therefore, all 10 design tasks may not always be necessary for every WebApp interface.

Interface Design Preliminaries

A key tenet of the WebE process model is this: *You'd better understand the problem before you attempt to design a solution.* In the case of interface design, understanding the problem means understanding (1) the people (end users) who will interact

with the system through the interface, (2) the tasks that end users must perform to do their work, (3) the content that is presented as part of the interface, and (4) the environment in which these tasks will be conducted. In the sections that follow, we examine each of these elements of interface analysis[5] with the intent of establishing a solid foundation for the design tasks that follow.

How Do We Understand the Characteristics of WebApp Users?

The phrase *user interface* is probably all the justification needed to spend some time understanding the user before worrying about technical matters. Each user has a mental image of the WebApp that may be different from the mental image developed by other users. In addition, the user's mental image may be vastly different from your design model. The only way that you can get the mental image and the design model to converge is to work to understand the users themselves as well as how these people will use the system.[6] Information from a broad array of sources can be used to accomplish this:

User interviews. The most direct approach is for members of the WebE team to meet with end users to better understand their needs, motivations, work culture, and a myriad of other issues. This can be accomplished in one-on-one meeting or through focus groups.

Sales input. Salespeople can meet with the customer and users on a regular basis and can gather information that will help the WebE team to categorize users and better understand their requirements.

Marketing input. Market analysis can be invaluable in the definition of market segments and an understanding of how each segment might use the software in subtly different ways.

Support input. Support staff can have discussions with users on a daily basis. They are the most likely source of information on what works and what doesn't, what users like and what they dislike, what features generate questions, and what features are easy to use.

The following set of questions (adapted from [Hac98]) will help you to better understand the users of a WebApp in a business setting:

- Are users trained professionals, technicians, clerical or manufacturing workers?

5 Some or all of these activities may have already been conducted as part of the communication activity (Chapter 4) or analysis modeling (Chapter 7).

6 Much, if not all, of the information about users and their needs should really be collected during requirements gathering and analysis. If these Web engineering actions have been performed properly, you will have enough information to understand the user. However, there are times when you must derive this information.

- What level of formal education does the average user have?
- Are the users capable of learning from written materials, or have they expressed a desire for classroom or mentor-based training?
- Are users expert typists or keyboard-phobic?
- What is the age range of the user community?
- Will the users be represented predominately by one gender?
- How are users compensated for the work they perform?
- Do users work normal office hours, or do they work until the job is done?
- Is the WebApp to be an integral part of the work users do, or will it be used only occasionally?
- What is the primary spoken language among users? What domain-specific language is used?
- What are the consequences if a user makes a mistake using the system?
- Are users experts in the subject matter that is addressed by the system?
- Do users want to know about the technology that sits behind the interface?

The answers to these and similar questions will allow you to understand who the end users are, what is likely to motivate and please them, how they can be grouped into different user classes or profiles, what their mental models of the system are, and how the user interface must be characterized to meet their needs.

How Do We Elaborate the Content Objects That Are Identified?

An interface designer also examines usage scenarios and other information obtained from the user and extracts the content objects that are manipulated by the tasks that the user performs. These objects can be categorized into classes. Attributes of each class are defined, and an evaluation of the actions applied to each object provide you with a list of operations.

In Chapter 7, we defined a data tree (Figure 7.5) for a **SafeHomeAssured.com** content object called **Component.** If you have responsibility for interface design you would examine the content object and use the description of **Component** to help in the layout of the interface. You should recognize that all users will have an interest in the object **Component** and will view it as a product specification for some component of *SafeHome*. Key content that must be presented via the interface include the product name, serial number, part category, description, price, photo, technical description, and other related information.

You then create the layout shown in Figure 9.3. This Web page is the template for each product specification. Note that the form of the page remains identical to

FIGURE 9.3

Security products page layout.

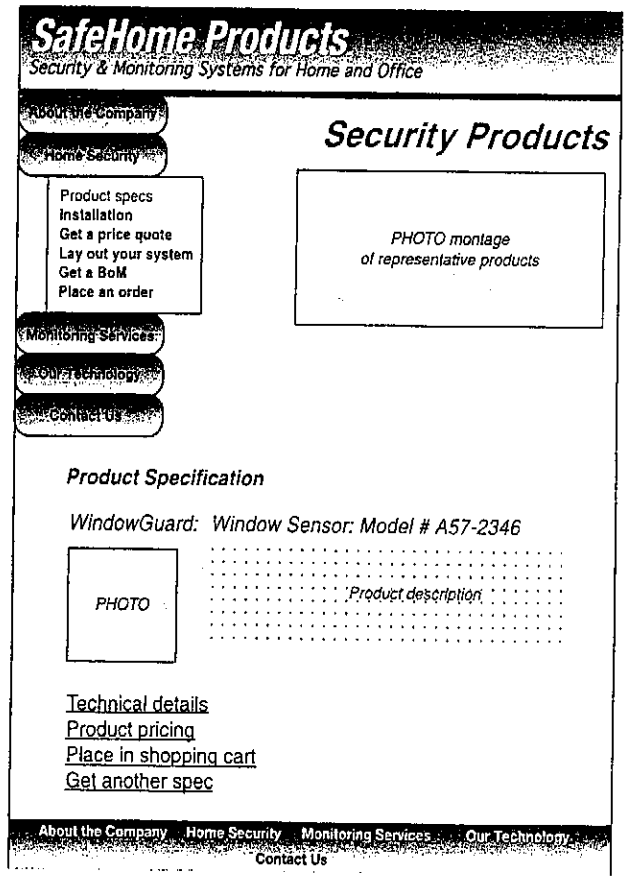


Figure 9.1, though navigation menus[7] are expanded to reflect the tasks that will be applied to the **Component** (product specification) content object. The product name, category, part number, description, and photo are represented as shown in the figure.

Applying the design principles discussed earlier in this chapter, the interface design encourages ease of understanding and use. For example, rather than

7 You have chosen to use pull-down navigation menus for subfunctions (e.g., "Product specs," "Installation," "Get a price quote"), while always allowing the user to have access to major functions that were presented on the home page.

providing technical details directly, the user can navigate to them (not every user has an interest and their inclusion would make the page very busy) via the "Technical details" navigation link. Other product specification-related navigation links—"Product pricing," "Place in shopping cart," and "Get another spec"— are shown in the lower left-hand corner of the page.

What Tasks Do the Users Perform?

In Chapters 4 and 7, we emphasized the need to obtain answers to the following questions:

- What work will the user perform in specific circumstances?
- What tasks and subtasks will be performed as the user does the work?
- What specific problem domain objects will the user manipulate as work is performed?
- What is the sequence of work tasks—the workflow?
- What is the hierarchy of tasks?

Hopefully, these questions have been answered before design commences.

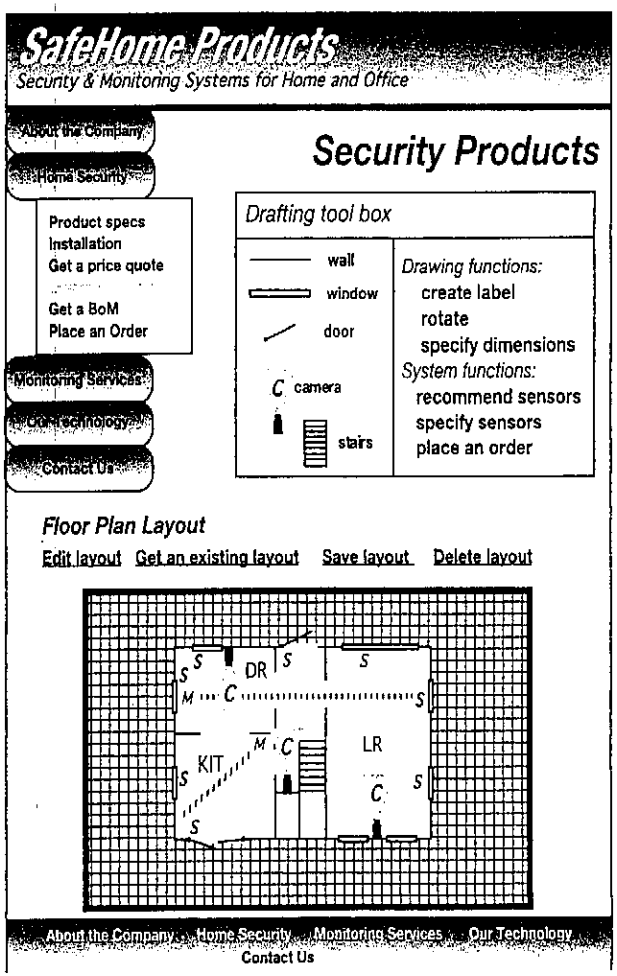
In earlier chapters we noted that the use case describes the manner in which an actor (in the context of user interface design, an actor is always a person) interacts with a system. When used as part of task analysis, the use case is developed to show how an end user performs some specific work-related task. In most instances, the use case is written in an informal style (a simple paragraph) in the first person.

As a result of requirements gathering (Chapter 4), the **SafeHomeAssured.com** WebE team defined an increment named *Develop a layout from the space to be monitored.* The interface designer reviews the use case for this increment:

Use case: *Develop a layout for the space to be monitored*
Actor: Any user
Description: I want to configure a security system by first representing the layout of a "space" (i.e., house, office/retail space) in which the security sensors and monitoring devices are to be installed. To accomplish this, I must be able to use a "drafting tool box" that will allow me to draw walls, windows, and doors for the floor plan. I must be able to specify dimensions for walls, doors, and windows; orient them properly; and represent security sensors and monitoring devices as required. Each of these drawing elements (walls, windows, doors, sensors, monitoring devices) must be capable of being moved and rotated. I want to be able to have SafeHomeAssured.com analyze the floor plan layout I develop and recommend sensor and monitoring device placement. I want to save a floor plan by name, retrieve it at a later time, edit it, and delete it. Once the floor plan and sensor placement are complete, I want to be able to order the configuration of sensors and monitoring devices. All of these actions must be secure.

This use case provides a basic description of the interface for the floor plan layout. From it, you can extract tasks, objects, and the overall flow of the interaction. In addition, extended WebApp features that might please the user can also be conceived. For example, multifloor spaces were not explicitly mentioned in the use case but are essential for a complete floor plan layout capability. You recognize this and provide additional functionality. The resultant interface design for the floor plan layout is shown in Figure 9.4.

FIGURE 9.4
Security products floor plan layout interface design.



How Do We Elaborate the Tasks That Are Identified?

Task analysis can be applied in two ways. As we have already noted, a WebApp is often used to replace stand-alone software or a manual or semiautomated activity. To understand the tasks that must be performed to accomplish the goal of the activity, you must understand the tasks that humans currently perform (when using the existing approach) and then map these into a similar (but not necessarily identical) set of tasks that are implemented in the context of the WebApp user interface. Alternatively, you can study the known requirements (including use cases) for the WebApp and derive a set of user tasks.

Regardless of the overall approach to task analysis, as a designer you must define and classify tasks. One approach is stepwise elaboration. For example, let's reconsider the use case *Develop a layout for the space to be monitored.* This use case explicitly cites a number of major tasks and implies a larger number of subtasks. Each major task can be elaborated into subtasks that are implemented as part of the interface. For example, consider the following use case excerpt:

> **I must be able to use a "drafting tool box" that will allow me to draw walls, windows, and doors for the floor plan. I must be able to specify dimensions for walls, doors, and windows; orient them properly; and represent security sensors and monitoring devices as required. Each of these drawing elements (walls, windows, doors, sensors, monitoring devices) must be capable of being moved and rotated.**

Referring to Figure 9.4, a **DraftingToolBox** (a content object) is implemented in the interface design. Within **DraftingToolBox,** drawing subfunctions—*Create a label, Rotate,* and *Specify dimensions* are noted explicitly. Functions that enable the user "to draw walls, windows, and doors for the floor plan" must be implemented but are not explicitly named in the interface. To accomplish the drawing function, **Wall, Window,** and **Door** objects are dragged and dropped to form the floor plan grid toward the bottom of the interface layout.

Other system functions—*Recommend sensors* and *Specify sensors*—imply a collection of complex functional components that implement specialized algorithms to perform those functions. These functional components are initiated when the user selects the appropriate link within **DraftingToolBox.**

The "Place an order" link allows the user to navigate to the e-commerce section of the **SafeHomeAssured.com** WebApp. However, internal data structures must be created to store the list of all sensors and monitoring devices that have been manually specified or automatically recommended. This data structure will then be used to develop a bill of materials and a price quote that are necessary to complete an order.

The interface design should be reviewed against the use case to ensure that each explicit and implicit user task is adequately addressed. As design work proceeds, the model of the interface should accommodate each of the tasks in a way that is consistent with the *user model* (the profile of a typical WebApp user) and *system perception* (what the user expects from the WebApp).

How Do We Design for Different Users with Different Roles?

When a number of different users, each playing different roles, makes use of a user interface, it is sometimes necessary to go beyond task analysis and object elaboration and apply *workflow analysis.* This technique allows you to understand how a work process is completed when several people (and roles) are involved. Consider a drug company that intends to implement a system that fully automates the process of prescribing and delivering prescription drugs. The entire process[8] will revolve around a Web-based application that is accessible by physicians (or their assistants), pharmacists, and patients. Workflows can be represented effectively with a UML swimlane diagram (a variation on the activity diagram)—particularly where there are multiple users or other entities participating in the interaction.

We consider only a small part of the work process—the situation that occurs when a patient asks for a refill. Figure 9.5 presents a swimlane diagram that indicates the tasks and decisions for each of the three roles noted in the preceding paragraph. This information may have been elicited via interview or from use cases written by each actor. Regardless, the flow of events (shown in the figure) enables the interface designer to recognize a number of key interface characteristics:

1. Each user implements different tasks via the interface; therefore, the look and feel of the interface designed for the patient will be different from the one designed for pharmacists or physicians.
2. The interface design for pharmacists and physicians must accommodate access to and display of information from secondary information sources (e.g., access to inventory for the pharmacist and access to information about alternative medications for the physician).
3. Many of the activities noted in the swimlane diagram can be further elaborated using task analysis and/or object elaboration (e.g., *Fills prescription* could imply a mail-order delivery, a visit to a pharmacy, or a visit to a special drug distribution center).

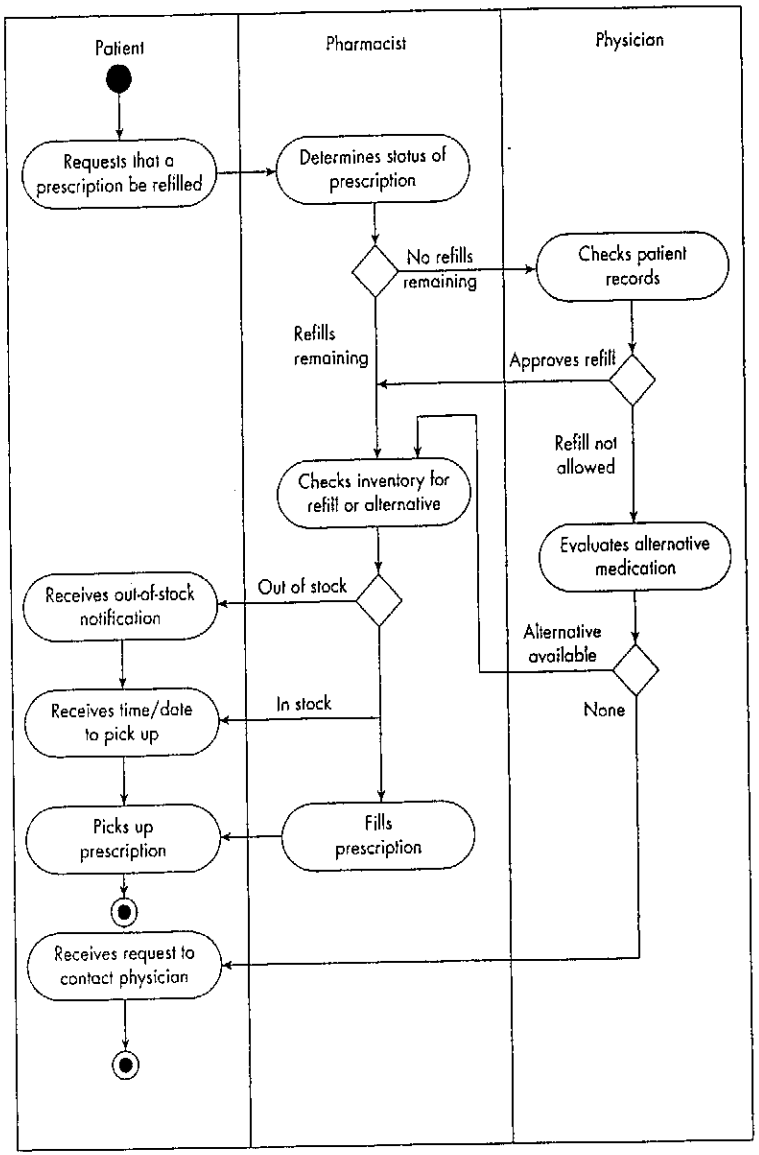
As the interface is analyzed, the process of elaboration continues. Once the workflow has been established, a task hierarchy can be defined for each user type. The hierarchy is derived by a stepwise elaboration of each task identified for the user. For example, consider the user task *Requests that a prescription be refilled.* The following task hierarchy is developed:

---

8 This example has been adapted from [Hac98].

FIGURE 9.5

Swimlane diagram for prescription refill function.



Requests that a prescription be refilled

- *Provide identifying information*
  - *Specify name*
  - *Specify userID*
  - *Specify PIN and password*
- *Specify prescription number*
- *Specify date and time refill is required*

To complete the *Requests that a prescription be refilled* tasks, three subtasks are defined. One of these subtasks, Provide identifying information, is further elaborated in three additional sub-subtasks.

How Is Content Integrated into the Interface Description?

The user tasks we identified in the preceding section ("How Do We Design for Different Users with Different Roles?") lead to the presentation of a variety of different types of content. For modern applications, display content can range from character-based reports (e.g., a spreadsheet), graphical displays (e.g., a histogram, a 3D model, a picture of a person), or specialized information (e.g., audio or video files). Content objects may be (1) generated by components (unrelated to the interface) in other parts of the WebApp, (2) acquired from data stored in a database that is accessible from the application, or (3) transmitted from systems external to the WebApp.

The format and aesthetics of the content (as it is displayed by the interface) must be considered as part of interface design. Some of the questions you should consider are as follows:

- Are different types of data assigned to consistent geographic locations on the screen (e.g., do photos always appear in the upper right-hand corner)?
- Can the user customize the screen location for content?
- Is proper on-screen identification assigned to all content?
- If a large report is to be presented, how should it be partitioned for ease of understanding?
- Will mechanisms be available for moving directly to summary information for large collections of data?
- Will graphical output be scaled to fit within the bounds of the display device that is used?
- How will color be used to enhance understanding?
- How will error messages and warnings be presented to the user?

As each question is answered, the design requirements for content presentation are established.

Interface Design Steps

Once design preliminaries have been completed, all tasks (or objects and actions) required by the end user have been identified in detail and the interface design activity commences. Interface design, like all WebE design, is an iterative process. Each user interface design step occurs a number of times, each elaborating and refining information developed in the preceding step.

Although many different user interface design models have been proposed, all suggest some combination of the following steps:

1. Using information developed during design preliminaries, define interface objects and actions (operations).
2. Define events (user actions) that will cause the state of the user interface to change. Model this behavior.
3. Depict each interface state as it will actually look to the end user. Create an aesthetic, and lay out all navigation mechanisms, content objects, and related information.
4. Indicate how the user interprets the state of the system from information provided through the interface.

In some cases, you may begin with sketches of each interface state (i.e., what the user interface looks like under various circumstances) and then work backward to define objects, actions, and other important design information. Regardless of the sequence of design tasks, you should: (1) always follow the design principles and guidelines discussed earlier in this chapter, (2) model how the interface will be implemented, and (3) consider the environment (e.g., display technology, operating system, development tools) that will be used.

How Are Interface Objects and Actions Translated into a Layout?

Once interface objects and actions have been defined and elaborated iteratively, they are categorized by type. Target, source, and application objects are identified. A *source object* (e.g., a report icon) is dragged and dropped onto a *target object* (e.g., a printer icon). The implication of this action is to create a hard-copy report. An *application object* represents application-specific data that are not directly manipulated as part of screen interaction. For example, a mailing list is used to store names for a mailing. The list itself might be sorted, merged, or purged (menu-based actions), but it is not dragged and dropped via user interaction.

SafeHome

Designing a Layout

The scene: The Web engineering area at *CPI Corporation*

The players: Two members of the **SafeHome-Assured.com** team working as a pair, who are designing the page layout for the use case, *Access camera surveillance via the Internet*

The conversation:

Team member 1: This use case we have from analysis doesn't provide us with a lot of detail.

Team member 2: Agreed, but it's probably enough to get started. We'll have to elaborate as required.

Team member 1: Will marketing be around if we have questions?

Team member 2: They better be. They're part of the team.

[Team member 1 draws a sketch for the camera surveillance page (Figure 9.6).]

Team member 1: Whaddaya think?

Team member 2: Well . . . I think we need to make some mods. First, look at this use case. They explicitly want to select the camera, sensors, etc., from the floor plan. I don't see a floor plan.

Team member 1: I thought it would complicate the page, and besides, it'll be more work to implement. I used a list of camera and other sensors . . . we could add their location . . . same thing.

Team member 2: Nah, it isn't, and besides, we have to work with the use cases we have.

Team member 1 (sighs): Alright, you have a point. Then we've got to parse the use case and pick out key content objects and the operations that we'll need to manipulate them and then, what?

Team member 2: We need to be sure that all the content objects appear within the interface design and that all the operations are implemented in some manner.

Team member 1: You know, I just noticed that we're not providing any way to control camera functions, you know, on/off, pan/zoom.

Team member 2: That's one of the ops we have to implement.

[Both team members work on a revised sketch that is a precursor to Figure 9.7.]

When you are satisfied that all important objects and actions have been defined (for one design iteration), perform the screen layout. Like other interface design activities, screen layout is an iterative process in which graphical design and placement of icons, definition of descriptive screen text, design and placement of navigation mechanisms, specification and titling for windows, and definition of major and minor menu items are conducted. If a real-world metaphor is appropriate for the application, it is specified at this time and the layout is organized in a manner that complements the metaphor.

To provide a brief illustration of the design steps noted previously, we revisit a **SafeHomeAssured.com** function called *Access camera surveillance via the Internet*. The stakeholder who takes on the role of the **HomeOwner** actor might write the following use case:

Use case: *Access camera surveillance via the Internet*
Actor: HomeOwner

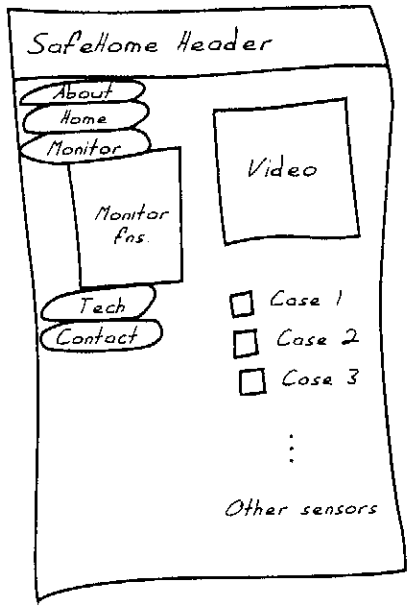


FIGURE 9.6 Preliminary sketch for the *Camera Surveillance* interface.

Narrative: If I'm at a remote location, I can use any PC with appropriate browser software to log on to the ***SafeHomeAssured.com*** website. I enter my user ID and two levels of passwords, and once I'm validated, I have access to all functionality for my installed *SafeHome* system. To access a specific camera view, I select "Surveillance" from the menu options displayed. I then select "Pick a camera," and the floor plan is displayed. I then select the camera that I'm interested in. Alternatively, I can look at thumbnail snapshots from all cameras simultaneously by selecting "All cameras" as my viewing choice. Once I choose a camera, I select "View" and a camera view appears in a viewing window that is identified by the camera ID. I can pan and zoom the camera I have selected. If I want to switch cameras, I again select "Pick a camera" and the original viewing window disappears. I then select the camera that I'm interested in. A new viewing window appears.

Based on this use case, the following homeowner tasks, objects, and data items are identified:

- *Accesses* the **SafeHome** system
- *Enters* an **ID** and **Password** to allow remote access
- *Displays* **FloorPlan** and **SensorLocations**
- *Displays* **VideoCameraLocations** on floor plan
- *Selects* **VideoCamera** for viewing
- *Views* **VideoImages** (four frames per second)
- *Pans* or *zooms* the **VideoCamera**

Objects (boldface) and actions (italics) are extracted from this list of user tasks.

A preliminary design layout of the screen layout for video monitoring is created (Figure 9.7).[9] To invoke the video image, a video camera location icon, C, located in the floor plan displayed in the monitoring window is selected. In this case, a camera location in the living room (LR) is selected. The video image window appears, displaying a video from the camera located in the living room. The zoom and pan control slides are used to control the magnification and direction of the video image. To select a view from another camera, the user selects "Pick a camera" from the menu and a new video window replaces the original.

The preliminary design layout for the interface shown in Figure 9.7 would have to be supplemented with additional figures (sketches) representing the expansion of each menu item within the menu bar, indicating what actions are available for each monitoring services mode (state). After review of the preliminary design, two issues arise:

- The zoom and pan sliders should be placed immediately below the video window.
- Stakeholders would like to have the capability to record the streaming video, and replay it.[10]

Given these issues, the preliminary design is revised as shown in Figure 9.8.

What About the Design of Navigation Mechanisms for the Interface?

As we noted earlier in this chapter, the objectives of a WebApp interface are to: (1) establish a consistent window into WebApp content and functionality, (2) guide the user through a series of interactions with the WebApp, and (3) organize the navigation options and content available to the user. To achieve a consistent interface, you should first use aesthetic design (considered in the section "Aesthetic Design" that follows) to establish a coherent look for the interface. This encompasses many characteristics, but must emphasize the layout and form of navigation mechanisms. To guide user interaction, you may draw on an appropriate metaphor[11] that enables the user to gain an intuitive understanding of the interface. To implement navigation options, you can select from one of a number of interaction mechanisms:

9 Note that this differs somewhat from the implementation of these features in earlier chapters. This might be considered a first-draft design and represents one alternative that might be considered.

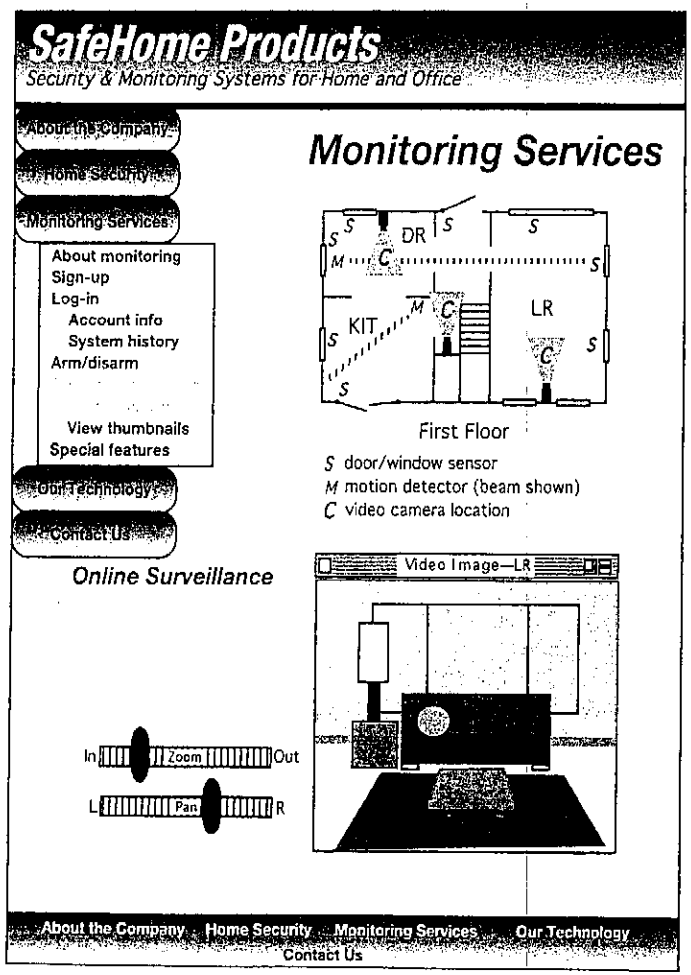
10 This represents a new feature for the WebApp and would have to be negotiated to ensure that increment delivery dates can be maintained or adjusted.

11 In this context, a *metaphor* is a representation (drawn from the user's real-world experience) that can be modeled within the context of the interface. A simple example might be a slider switch that is used to control the auditory volume of an .mpg file.

- **Navigation menus.** Key word menus (organized vertically or horizontally) list key content and/or functionality. These menus may be implemented so that the user can choose from a hierarchy of subtopics that is displayed when the primary menu option is selected. Referring to Figure 9.8, the key word menus are organized vertically on the left of the Web page. Selecting a major function (e.g., monitoring services) causes a submenu to appear as

FIGURE 9.7

Preliminary design layout for the monitoring services interface.

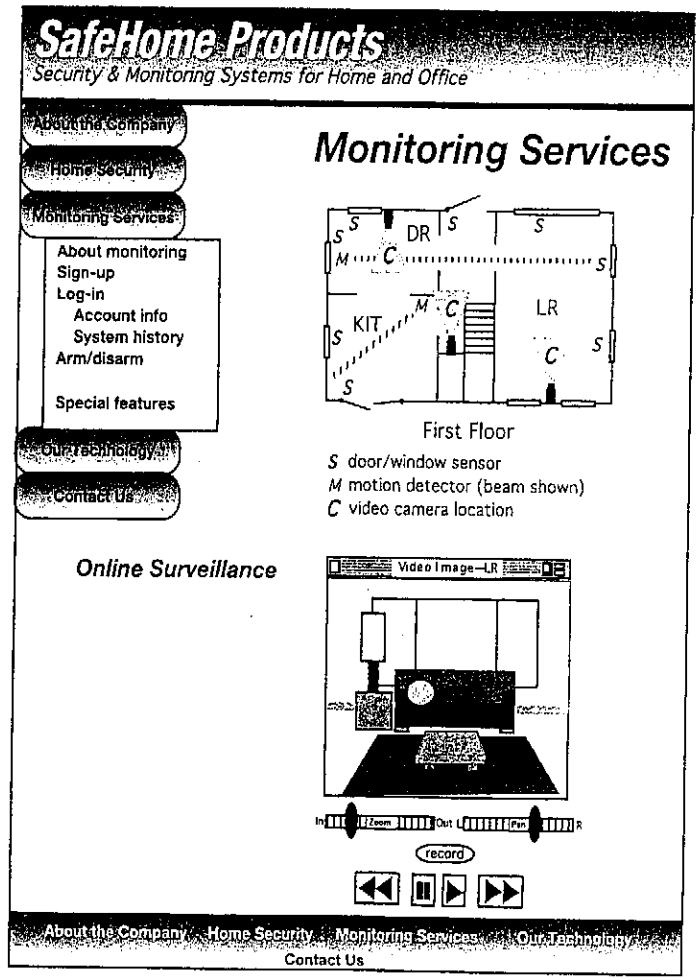


shown. The subfunctions contained within the menu can then be selected to meet the user's needs.

- **Graphic icons.** Buttons, switches, and similar graphical images enable the user to select some property or specify a decision. Referring again to Figure 9.8, the slide switches and recorder controls for the video image are graphic icons that are consistent with a video metaphor.

FIGURE 9.8

Monitoring services online surveillance revised design layout.



- **Graphic images.** Graphical representations are selectable by the user and implement a links to content objects or WebApp functionality. In Figures 9.7 and 9.8, the floor plan layout contains a variety of graphical images (e.g., the camera icon) that can be selected by the user.

One or more of these control mechanisms should be provided at every level of the content hierarchy.

It is important to distinguish between the layout of navigation mechanisms (an interface design issue) and the design of navigation for the WebApp. Navigation design, discussed in Chapter 10, focuses on the semantics of the navigational flow between content objects and for all WebApp functions. Interface design implements the semantics using a syntax of navigation mechanisms (menus, icons, images).

Why Is Interface Consistency So Important?

A well-designed WebApp interface should be both internally and externally consistent. As we have already noted, *internal consistency* demands that the aesthetics of each Web page have the same look and feel, that navigation mechanisms are used in a uniform manner, that content objects have the same fundamental design characteristics, and that input and output from WebApp functions exhibit a regularity that is predictable and pleasing. Internal consistency allows the user to develop an interactive rhythm that leads to high ratings for usability.

But internal consistency is not enough. The WebApp must also exhibit *external consistency*. Like it or not, the typical visitor to a WebApp will not spend much time there. It's likely that the visitor will arrive after visiting other websites and will leave to travel to still other WebApps. As Nielsen [Nie00] notes: ". . . users feel that they are using the Web as a whole rather than any specific site." It is for this reason that your WebE team must abide by current interface design conventions, making your WebApp's interface conventions consistent with those used across the Web. If you don't (after all, each of us likes to be creative and think outside the box), you risk alienating visitors who may not have the time or the inclination to learn a new interaction paradigm.

Aesthetic Design

The best WebApp interfaces are efficient in their presentation of content and function, but they should also be aesthetically pleasing. Aesthetic design, also called *graphic design*, is an artistic endeavor that complements the technical aspects of both interface design and content design. Without it, a WebApp may be useful, but

unappealing. With it, a WebApp draws its users into a world that embraces them on a visceral, as well as an intellectual level.

But what is aesthetic? There is an old saying, "beauty exists in the eye of the beholder." This is particularly appropriate when aesthetic design for WebApps is considered. To perform effective aesthetic design, we again return to the user hierarchy developed as part of the analysis model (Chapter 7) and ask, "Who are the WebApp's users, and what WebApp look do they desire?"

How Do We Create an Aesthetically Pleasing Layout?

Every Web page has a limited amount of "real estate" that can be used to support nonfunctional aesthetics, navigation features, informational content, and user-directed functionality. The development of this real estate is planned during aesthetic design.

Like all aesthetic issues, there are no absolute rules when screen layout is designed. However, a number of general layout guidelines are worth considering:

- **Don't be afraid of white space.** It is inadvisable to pack every square inch of a Web page with information. The resulting clutter makes it difficult for the user to identify needed information or features and creates visual chaos that is not pleasing to the eye. Referring back to Figure 9.3, the designer has chosen to provide a navigation link to content describing technical details rather than incorporating this content on the existing page. The reason is to keep the page from becoming too busy and preserve white space.
- **Emphasize content.** After all, that's the reason the user is there. Nielsen [Nie00] suggests that the typical Web page should be 80 percent content with the remaining real estate dedicated to navigation and other features. Referring back to Figure 9.1, the **SafeHomeAssured.com** home page is almost all content, providing the user with a picture of what *SafeHome* products are about.
- **Organize layout elements from top-left to bottom-right.** The vast majority of users will scan a Web page in much the same way as they scan the page of a book—top-left to bottom-right. If layout elements have specific priorities, high-priority elements should be placed in the upper-left portion of the page real estate.
- **Group navigation, content, and functionality geographically within the page.** Humans look for patterns in virtually all things. If there are no discernable patterns within a Web page, user frustration is likely to increase

(due to unnecessary searching for needed information). The interface design (e.g., Figure 9.3) for **SafeHomeAssured.com** places all major functions in the upper left-hand corner and has all content presented left to right and top to bottom.

- **Don't extend your real estate with the scrolling bar.** Although scrolling is often necessary (especially once you drill down to the detailed content pages), most studies indicate that users would prefer not to scroll. It is better to reduce page content or to present necessary content on multiple pages. The interface designer for the **SafeHomeAssured.com** WebApp violates this rule (it may be necessary to scroll to certain functionality or content), arguing that necessary content overrides a desire to limit scrolling.
- **Consider resolution and browser window size when designing the layout.** Rather than defining fixed sizes within a layout, the design should specify all layout items as a percentage of available space [Nie00].
- **Design the layout for freedom of navigation.** The generic layout of all WebApp pages should assume that the user will navigate to the page in ways that are not expected (e.g., via a direct link from a search engine). The layout should be designed to accommodate unpredictable arrival without confusion (on the part of the user). It is also important to ensure that a user is not allowed to navigate to a secure page without first passing through security validation. For example, if a visitor tried to navigate directly to the **SafeHomeAssured.com** monitoring services page shown in Figure 9.8, the visitor would be redirected automatically to the *log-in* page.
- **Don't assume that the layout will be consistent across different display devices and browsers.** Design the layout so that it is effective on both large and small displays, using features that are properly translated on the majority of popular browsers.
- **If you use photos, make them small format with the option to enlarge.** Large JPEG files can take time to download. Most users will be satisfied with a thumbnail photo as long as they have the option to look at a larger version.
- **If you want a cohesive layout, look, and feel across all WebApp pages, use a cascading style sheet (CSS).** A CSS allows you to specify one look and feel (e.g., font type, size, and style) across all Web pages. Just as important, the CSS lets you modify the look and feel across all pages by making changes to only one file.

Nielson [Nie00] suggests an interesting design exercise when he states: ". . . go through your design elements and remove them one at a time. If your design works without a certain element, kill it. Simplicity always wins over complexity . . ."

SAFEHOME

Reviewing the Aesthetic Design

The scene: The Web engineering area at CPI Corporation

The players: Two members of the **SafeHomeAssured.com** team working as a pair, who are conducting a pair walkthrough on the aesthetic design of the product specification page (Figure 9.3)

The conversation:

Team member 1 (looking at Figure 9.3): I'm generally pleased with this. It seems to work.

Team member 2: I've been thinking about it, and we have some aesthetic problems.

Team member 1: What problems? Functions are delineated nicely top-left. We use plenty of white space. The product info is easy to find on the page.

Team member 2 (gently interrupting): I'm not so sure that the product info is easy to find. Users working on a small screen will have to scroll to see it. Worse, they'll have to scroll to see the links to technical details, product pricing, and so on.

Team member 1 (thinking for a moment): Yeah, you're right about the scrolling. What if we get rid of the photo montage and move the product specification content upward on the page?

Team member 2: That'd work, but we have to be sure we don't crowd stuff. Remember, white space. [She pauses for a moment.] There's something else.

Team member 1 (interrupting): Before you get to it, I just noticed something. We never provided a mechanism for increasing the size of the component photo. I'll note it so we can make the change. What were you going to say?

Team member 2: Aesthetically, I think we can do a better job with the links to technical details, product pricing, etc. Right now, they're very basic and visually unappealing, don't you think?

Team member 1: Actually, you're right, and the fact that they're low on the page and removed from the other functions is not the best. Maybe what we need to do is create a third level of functionality that pops up once "Product Specs" is chosen.

Team member 2: I like that. All functions appear upper-left. But we have to be sure that the functional hierarchy doesn't become too busy.

Team member 1: It's doable, but the functional hierarchy has to appear on all pages where it's needed—consistency and all that.

Team member 2: We've got a few changes to make.

Team member 1: Piece of cake.

What Leads to an Effective Graphic Design?

Graphic design considers every aspect of the look and feel of a WebApp. The graphic design process begins with layout and proceeds into a consideration of global color schemes, text types, sizes, and styles; the use of supplementary media (e.g., audio, video, animation); and all other aesthetic elements of an application.

A comprehensive discussion of graphic design issues for WebApps is beyond the scope of this book. You can obtain design tips and guidelines from many websites that are dedicated to the subject. For example,

Design & Publishing eZine	**www.graphic-design.com**
Grantastic Designs	**www.grantasticdesigns.com**
Web Page Design for Designers	**www.wpdfd.com**

or from one or more print resources (e.g., [Bag01], [Clo01], or [Hei02]).

INTERACTION DESIGN

Well-Designed Websites

Sometimes, the best way to understand good WebApp design is to look at a few examples. In his article, "The Top Twenty Web Design Tips," Marcelle Toor (www.graphic-design.com/Web/feature/tips.html) [Too02] suggests the following websites as examples of good graphic design.

Primo Plastics, www.primo.com. A design firm headed by Primo Angeli

Workbook Assignment Portfolios, www.workbook.com. A site that showcases work by illustrators and designers

River of Song, www.pbs.org/riverofsong. A television series for public TV and radio about American music

Design Office of RKD, www.RKDINC.com. A design firm with an online portfolio and good design tips

Creative Hotlist, www.commarts.com/career/index.html. *Communication Arts* magazine, a trade periodical for graphic designers and a good source for other well-designed sites

BTD/Beth Tondreau Design, www.btdnyc.com. A design firm headed by Beth Toudreau

Usability

In an insightful paper on usability, Larry Constantine [Con95] asks a question that has significant bearing on the subject: "What do users want, anyway?" He answers this way:

> What users really want are good tools. All software systems, from operating systems and languages to data entry and decision support applications, are just tools. End users want from the tools we engineer for them much the same as we expect from the tools we use. They want systems that are easy to learn and that help them do their work. They want software that doesn't slow them down, that doesn't trick or confuse them, that doesn't make it easier to make mistakes or harder to finish the job.

Constantine argues that usability is not derived from aesthetics, state-of-the-art interaction mechanisms, or built-in interface intelligence. Rather, it occurs when the architecture of the interface fits the needs of the people who will be using it.

A formal definition of usability is somewhat illusive. Donahue and his colleagues [Don99] define it in the following manner: "Usability is a measure of how well a computer system . . . facilitates learning; helps learners remember what they've learned; reduces the likelihood of errors; enables them to be efficient, and makes them satisfied with the system."

The only way to determine whether "usability" exists within a WebApp you are building is to conduct a usability review once the interface design has been established. Watch users interact with the WebApp design prototype, and answer the following questions [Con95]:

- Is the WebApp usable without continual help or instruction?
- Do the rules of interaction and navigation help a knowledgeable user work efficiently?

- Do interaction and navigation mechanisms become more flexible as users become more knowledgeable?
- Has the WebApp been tuned to the physical and social environment in which it will be used?
- Are users aware of the state of the WebApp? Do users know where they are at all times?
- Is the interface structured in a logical and consistent manner?
- Are interaction and navigation mechanisms, icons, and procedures consistent across the interface?
- Does the interaction anticipate errors and help users correct them?
- Is the interface tolerant of errors that are made?
- Is the interaction simple?

If each of these questions is answered "yes," it is likely that usability has been achieved.

Among the many measurable benefits derived from a usable system are increased sales and customer satisfaction, competitive advantage, better reviews in the media, better word of mouth, reduced support costs, improved end-user productivity, reduced training costs, reduced documentation costs, and reduced likelihood of litigation from unhappy customers [Don99].

Design Issues

As the design of a WebApp interface evolves, five common design issues almost always surface: system response time, user help facilities, error information handling, accessibility, and internationalization. Unfortunately, many designers do not address these issues until relatively late in the design process (sometimes the first inkling of a problem doesn't occur until an operational increment has been deployed). Unnecessary iteration, project delays, and customer frustration often result. It is far better to establish each as a design issue to be considered at the beginning of interface design, when changes are easy (at least when compared with the difficulty in addressing them later in the development) and costs are low.

What Factors Affect Response Time and What Can We Do to Improve It?

System response time for WebApps is complicated by many factors: instantaneous load on the server, the bandwidth of the client-server connection, the processing speed of the client, the complexity of content and the functions that create it, and the capabilities of the browser. It should come as no surprise that response time is the primary complaint for many WebApps. In general, system response time is measured

from the point at which the user performs some control action (e.g., hits the return key or clicks a mouse) until the WebApp responds with desired output or action.

System response time has two important characteristics: duration and variability. If the duration of system response is too long, user frustration and stress is the inevitable result. *Variability* refers to the deviation from average response time, and in many ways, it is the most important response time characteristic. Low variability enables the user to establish an interaction rhythm, even if response time is relatively long. For example, a 1-second response to a command will often be preferable to a response that varies from 0.1 to 2.5 seconds. When variability is significant, the user is always off balance, always wondering whether something different has occurred behind the scenes.

Because so many factors affect response time, it is difficult for a designer to create a design (at any level) that guarantees a short, predictable (low variability) response. At the interface design level,[12] it is important to weigh specific requirements for the WebApp against its overall hardware, software, and communication characteristics. For example, if the WebApp is likely to service a very high number of users, the server configuration must be designed to accommodate the load; if the communication bandwidth for the average user is predicted to be relatively low, the use of large content objects (such as large photographs, graphics files, or video content) should be avoided when possible; if a processing function is extremely complex and compute intensive, it might be best to segment it into client-side and server-side functionality to maximize processing speed or, alternatively, to warn the user if a selection (of a processing function or content object) will require a significantly longer response or download time than normal. It is also worthwhile to provide a graduated time scale that provides the user with an indication of the percent completion of the operation. In extreme cases, the use of technologies such as AJAX can overcome some of the limitations of the Web and improve response times.

How Should We Design "Help" Facilities?

You should strive to create a WebApp interface that is intuitive and, therefore, completely self-explanatory. But almost every user of an interactive, computer-based system requires help now and then. Therefore, you should provide a set of obvious mechanisms for those in need of help. Once the user navigates to a "help" facility, a graduated solution context should be provided:

1. Most common problems can be answered with a *Frequently asked questions* (FAQ) facility. But many WebApps design this facility as an afterthought, making it extremely sparse and therefore useless. The FAQ should be categorized by topic, have a wide variety of common and not-so-common questions (that are updated regularly), and be detailed in its level of response.
2. An online *Help* facility should be provided for particularly complex websites. The user guide should have a detailed table of contents, a search facility, and an extremely detailed index.
3. A technical support link should be provided as a last resort.

A number of design issues [Rub88] must be addressed when an online help facility is developed:

- *Will help be available for all WebApp functions and at all times during system interaction?* Options include help for only a subset of all functions and actions or help for all functions.
- *How will the user request help?* Options include a "Help" menu item or a navigation icon (e.g., a stylized question mark).
- *How will help be represented?* Options include a separate window, a reference to a PDF document (less than ideal), or a one- or two-line suggestion produced in a fixed screen location.
- *How will the user navigate back to normal interaction?* Options include a return button displayed on the screen or the browser back button (less than ideal).
- *How will help information be structured?* Options include a "flat" structure in which all information is accessed through a key word or a layered hierarchy of information that provides increasing detail as the user proceeds into the structure via hyperlinks.

Purists will argue that none of this should be necessary if the interface designer does a proper job. But pragmatists (our position) recognize that even solid designs cannot anticipate every question and eliminate every problem.

How Should the Interface Handle Errors?

A well-designed WebApp interface should anticipate errors and help the user to avoid them. However, when errors do occur, the interface should provide an immediate, recognizable, and understandable notification of the error.

As an example of a commonly encountered scenario in which error avoidance and error notification are weak, consider the following:

> Prior to a download or order on a commercial WebApp, the user must check a box that indicates that he agrees to abide by licensing or other rules. The box is often small and is placed in a visually obscure location. Therefore, it is often missed and not checked. The user submits his "order," and nothing appears to happen. Puzzled, the user scans

12 It is important to note that system response time is also a function of content design, component-level design, and even architectural design.

the Web page and after some searching finds the following message (often in red small text): *Your request cannot be processed because an error has occurred or information is missing.*

There is no indication of what error occurred or what information is missing. The user must recheck his actions until he determines that the box has not been checked. This is a frustrating and time-consuming sequence of events.

In this scenario we encounter two problems. First, the interface designer failed to help the user avoid an error in the first place. By placing the checkbox in a visually obscure location, the designer makes the likelihood of an error (not checking the box) much higher. Second, the error message itself is weak. It should state: *You have forgotten to check the licensing box. Please do so now and resubmit.* No ambiguity and no wasted time.

In general, every error message or warning produced by a WebApp should have the following characteristics:

- The message should describe the problem in jargon that users can understand.
- The message should make it clear what options users have to rectify the problem and/or recover from the error.
- The message should indicate any negative consequences of the error (e.g., potentially corrupted data files) so that users can check to ensure that they have not occurred (or correct them if they have).
- The message should be accompanied by an audible or visual cue. That is, a beep might be generated to accompany the display of the message, or the message might flash momentarily or be displayed in a color that is easily recognizable as the error color.
- The message should be nonjudgmental. That is, the wording should never place blame on the user.

Because no one really likes bad news, few users will like an error message no matter how well designed. But an effective error message philosophy can do much to improve the quality of a WebApp and will significantly reduce user frustration when problems do occur.

What Is "Accessibility" and How Does It Apply to Interface Design?

As WebApps become ubiquitous, Web engineers must ensure that interface design encompasses mechanisms that enable easy access for those with special needs. *Accessibility* for users (and developers) who may be physically challenged is an imperative for moral, legal, and business reasons. A variety of accessibility guidelines (e.g., [W3C03]) provide detailed suggestions for designing interfaces that achieve

SAFEHOME

Design Issues as the Interface Evolves

The scene: The Web engineering area at CPI Corporation

The players: Two members of the **SafeHome-Assured.com** team working as a pair, who have completed the interface design for the *Camera Surveillance* increment

The conversation:

Team member 1 (looking at Figure 9.8 and examining related work products): You know, something occurred to me as I was finishing up the documentation of this.

Team member 2: What?

Team member 1: We forgot to specify the design of any increment-specific help facility or any error-handling features.

Team member 2 (grimacing): Jeez. You're right, but we have to move into deployment or we'll never deliver this increment on time.

Team member 1 (thinking a moment): I agree. Why don't we spend an hour making a list of error conditions that are specific to camera surveillance, and then we'll design and implement the error processing during construction.

Team member 2: Good strategy. We can do the same thing for camera surveillance "help." Then implement the help facility during construction.

Team member 1: Maybe we should check with marketing to determine how sophisticated they want the help facility to be. No one said much about this, and there's no mention of it in the use cases.

Team member 2: But we need a help facility, whether they mentioned it or not.

Team member 1: Agreed. But we can do a sophisticated one or something simple.

Team member 2: Since no one mentioned it, why don't we go with a simple FAQ for this increment. If we get feedback asking for more, we'll address it in another WebApp increment.

Team member 1: Given the time line, that's a reasonable approach. Let's go that way.

Team member 2: Good. Let's start making a list of possible error conditions.

varying levels of accessibility. Others [e.g., [App07] and [Mic07]) provide specific guidelines for "assistive technology" that addresses the needs of those with visual, hearing, mobility, speech, and learning impairments. Guidelines for developing accessible software can also be found at the IBM Human Ability and Accessibility Center, **www-03.ibm.com/able/access_ibm/disability.html**.

What Is "Internationalization" and How Does It Apply to WebApps?

Web engineers and their managers invariably underestimate the effort and skills required to create user interfaces that accommodate the needs of different locales and languages. Too often, interfaces are designed for one locale and language and then jury-rigged to work in other countries. The challenge for interface designers is to create "globalized" WebApps. That is, user interfaces should be designed to accommodate a generic core of functionality that can be delivered to all who use the WebApp. *Localization* features enable the interface to be customized for a specific cultural market.

A variety of internationalization guidelines (e.g., [IBM07]) are available to Web engineers. These guidelines address broad design issues (e.g., screen layouts may differ in various markets) and discrete implementation issues (e.g., different alphabets may create specialized labeling and spacing requirements). The *Unicode* standard [Uni07] has been developed to address the daunting challenge of managing dozens of natural languages with hundreds of characters and symbols.

Where We've Been . . . Where We're Going

The user interface is the window into WebApp content and function. In many cases, the interface molds a user's perception of the quality of the system. If the "window" is smudged, wavy, or broken, the user may reject powerful functions and content. In fact, a weak interface may cause an otherwise well-designed and solidly implemented application to fail.

A wide array of principles and guidelines help the designer to create an effective user interface. These can be summarized with the following rules: (1) place the user in control, (2) make interaction easy for the user, and (3) make the interface aesthetic and consistent. An organized design process should be conducted to achieve an interface that abides by these rules.

Interface design begins with a series of analysis tasks that define the profiles of various end users and delineate user tasks and actions with use cases, task and object elaboration, workflow analysis, and hierarchical task representations.

Once tasks have been identified, a set of interface objects and actions is extracted from the user scenarios that describe each task. This provides a basis for creation of a screen layout that depicts graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined.

In the design chapters that follow, we'll examine aspects of WebApp design that move slowly away from the user's world and toward the technical domain of the WebApp as a computer-based system. In Chapter 10, we'll start this journey by considering information design—the representation of content objects and the mechanisms that allow a user to navigate among them.

References

[App07] Apple Inc., *Accessibility,* 2007, **www.apple.com/accessibility/** (accessed August 6, 2007).

[Bag01] Baggerman, L., and S. Bowman, *Web Design That Works,* Rockport Publishers, 2001.

[Clo01] Cloninger, C., *Fresh Styles for Web Designers,* New Riders Publishing, 2001.

[Con95] Constantine, L, "What DO Users Want? Engineering Usability in Software," *Windows Tech Journal,* December 1995, **www.forUse.com** (accessed August 6, 2007).

[Dix99] Dix, A., "Design of User Interfaces for the Web," *Proc. of User Interfaces to Data Systems Conference,* September 1999, **www.comp.lancs.ac.uk/computing/users/dixa/topics/webarch/** (accessed August 6, 2007).

[Don99] Donahue, G., S. Weinschenck, and J. Nowicki, "Usability Is Good Business," Compuware Corp., July 1999, **http://www.half-tide.com/ResourcesFiles/UsabilityCost-BenefitPaper.pdf** (accessed August 6, 2007).

[Fit54] Fitts, P., "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement," *Journal of Experimental Psychology,* vol. 47, 1954, pp. 381–391.

[Hac98] Hackos, J., and J. Redish, *User and Task Analysis for Interface Design,* Wiley, 1998.

[Hei02] Heinicke, E., *Layout: Fast Solutions for Hands-on Design,* Rockport Publishers, 2002.

[IBM07] IBM, "Overview of Software Globalization," 2007, **www-128.ibm.com/developerworks/xml/library/x-i18n1.html** (accessed August 6, 2007).

[Mic07] Microsoft, *Accessibility,* 2007, **www.microsoft.com/enable/** (accessed August 6, 2007).

[Nie00] Nielsen, J., *Designing Web Usability,* New Riders Publishing, 2000.

[Nie96] Nielsen, J., and A. Wagner, "User Interface Design for the WWW," *Proc. CHI '96 Conf. on Human Factors in Computing Systems,* ACM Press, 1996, pp. 330–331.

[Rub88] Rubin, T., *User Interface Design for Computer Systems,* Halstead Press (Wiley), 1988.

[Tog01] Tognozzi, B., "First Principles," *askTOG,* 2001, **www.asktog.com/basics/firstPrinciples.html** (accessed August 6, 2007).

[Too02] Toor, M., "The Top Twenty Web Design Tips," 2002, **www.graphic-design.com/Web/feature/tips.html** (accessed August 6, 2007).

[Uni07] Unicode, Inc., *The Unicode Home Page,* 2007, **www.unicode.org/** (accessed August 6, 2007).

[W3C03] World Wide Web Consortium, *Web Content Accessibility Guidelines,* 2003, **www.w3.org/TR/2003/WD-WCAG20-20030624/** (accessed August 6, 2007).

10 Information Design

Content is at the heart of almost every WebApp. Indeed, for the vast majority of Web-based systems and applications, "Content is King."[1] Effective user interfaces, straightforward navigation, and rich functionality are all important, but many WebApps that have exhibited these characteristics have failed because they lacked meaningful content or because the content could not be located.

In Chapter 9 we examined user interface design. If you think about it for a moment, the user interface for most WebApps is a gateway to content, and content (along with the functions that manipulate it) is what enables the user to achieve the objectives identified during the communication activity (Chapter 4). In this chapter we consider how to provide the substance underneath the user interface—we will focus on *information design*.

There are really only three key issues[2] that should be considered when a WebE team performs information design:

- **Content.** What content is available?
- **Composition.** What views on that content do you wish to provide users?
- **Navigation.** How do the users gain access to those views?

There are different levels of abstraction at which you might consider these information design issues. For example, are you considering the high-level information architecture of a WebApp or the low-level detailed design of the specific content on each page? Are you designing the access strategy across major information categories or the specific navigation options within a narrow collection of content options?

The specific content requirements of a WebApp are first considered during the communication activity (Chapter 4) and are then elaborated and structured during analysis modeling (Chapter 7). During early elicitation sessions of the communication activity, Web engineers and other stakeholders identify the initial content objects (a named collection of related information). Elicitation also considers the user's overall goals in

1 See Bill Gates, "Content Is Where the Money Is on the Internet," 1996, www.microsoft.com/billgates/columns/1996essay/essay960103.asp (accessed January 23, 2006) and David Callan, "Content is King," undated, www.akamarketing.com/content-is-king.html (accessed August 7, 2007)

2 A fourth issue—how is the content managed?—will be considered in Chapter 12.

accessing the content and their background as it relates to interpreting the content. Finally, the communication activity results in the development of user scenarios or use cases that illustrate the use of the content.

During analysis modeling, members of the WebE team often construct a content model. The content model makes use of Web information exchange diagrams (Chapter 7) that can be used to help in the refinement of content objects. The result is a clear set of content specifications for the WebApp, but not necessarily the ways in which this information will be arranged within the WebApp.

In this chapter we consider how this content can be organized, accessed, and managed—the information design for the WebApp. We begin by examining the high-level information architecture—what its role is, what it contains, and how it can be developed. We'll then consider detailed information design and the specific navigation structures that are required to access content. Finally, we'll draw both the high-level information architectural design and the low-level navigation design approaches together into an information design workflow that can be applied to your WebApp projects.

Information Architecture

The overall goal of information design is to translate content requirements, usually in the form of a detailed set of content objects, into a specific information design for the WebApp. The overall strategy for information design usually combines both a bottom-up approach and a top-down approach.

Bottom-up information design is commonly used for small WebApps and simply involves designing and building actual Web pages and progressively linking these together so that the information views and the structure of the WebApp emerge organically. However, for larger WebApps a bottom-up approach can result in a solution that frustrates a user's attempts to locate information, presents that information (when it is found) out of context, and restricts easy accommodation of changes.

A *top-down information design* approach considers the overall organization, interrelationships, and structure of major content categories within a WebApp. This high-level design approach addresses aspects of the WebApp such as the overall structure of the application, the way in which the information is organized, and the ways in which users might access that information. This is the realm of the *information architecture*.

What Is an Information Architecture?

The term *information architecture* (IA) is widely used in the Web engineering community, and yet, although each person seems to know exactly what it means, definitions vary dramatically from person to person. In an effort to remove this lack of

consistency, a number of organizations (e.g., the *Information Architecture Institute*[3]), websites (e.g., IAWiki **www.iawiki.net/**), publications (e.g., *Boxes and Arrows*[4]), and conferences (e.g., the annual IA Summit[5]) are dedicated to information architecture. The role of an *information architect* is commonly encountered within a WebE team that is working on a large project.

An information architecture might be as simple as a site map that represents the basic WebApp navigational structure. Alternatively, an IA may be a detailed model that provides a comprehensive overview of the approach for structuring, managing, and accessing information within a WebApp. One definition[6] of an IA that captures the need to focus on the user's tasks is: *"The structural design of an information space to facilitate task completion and intuitive access to content"* [Ros02].

The IA provides a skeleton around which all information aspects of the WebApp are built. To elaborate on this a little further, let's change our focus for a moment and consider an example from the world of residential architecture.

Think about what a set of architectural plans for a house might actually show you. The plans will certainly indicate major structural elements of the house along with the materials from which the structure will be constructed. From the location of the major structural elements—room layout, stairs, walls, doors, windows—the internal space, and hence the effectiveness of the architectural solution, can be evaluated. The plans will also show how the house relates to its environment—the compass orientation of the house, its relationship to major environmental elements on the property (e.g., a pond or brook, rock outcroppings) and the slope of the land on which the house sits. Finally, the plans will show the location of the main access (e.g., driveway, walks) to the house and what other things border the property on which the house sits.

In other words, the architectural plans for a house show the basic structure and its context—those things that define the overall *shape* and usability of the residence. More detailed architectural information (e.g., plumbing and electrical layouts) can be incorporated as additional layers of information..

Because architectural plans show the relationship between the house and its surrounding environment, it is possible to evaluate whether the design is appropriate for its surroundings. The *utility* of the resultant house will be dependent not only on its structure but on how that structure relates to its environment.

An IA for a WebApp is (or should be) analogous to a set of architectural plans for a house. It should describe the basic information "structure" of the solution,

3 iainstitute.org
4 www.boxesandarrows.com/
5 ASIS&T Information Architecture Summit 2007, www.iasummit.org/
6 A good discussion on the various definitions for IA can be found at IAWiki, www.iawiki.net/DefiningTheDamnThing (accessed August 8, 2007).

but also position this within the overall information "landscape" in which the WebApp exists.

What Are the Elements of an Information Architecture?

In their seminal text on information architecture, Rosenfeld and Morville [Ros02] describe IA in terms of a Venn diagram containing three intersecting circles representing context, content, and users. In their introductory chapter they state:

> Some web sites provide logical structures that help us find answers and complete tasks. Others lack any intelligible organization and frustrate our attempts to navigate through them. We can't find the product we need; we can't locate the report we found last week; we're lost inside an online shopping cart. These web sites may remind us of buildings that fail: houses with flat roofs that leak, kitchens with no counter space, office towers with windows you can't open, and maze-like airports with misleading signs . . .
>
> However, as designers of web sites, we should not become trapped by the metaphor of building architecture . . . we'll also talk about information ecologies, knowledge economies, digital libraries, and virtual communities. We learn what we can from each analogy, and we leave the baggage behind.

They then go on to define IA:

> **In.for.ma.tion ar.chi.tec.ture** *n*
>
> 1. The combination of organization, labeling, and navigation schemes within an information system.
> 2. The structural design of an information space to facilitate task completion and intuitive access to content.
> 3. The art and science of structuring and classifying web sites and intranets to help people find and manage information.
> 4. An emerging discipline and community of practice focused on bringing principles of design and architecture to the digital landscape.

In other words, an IA is the high-level design that creates the structure of the information space to be used within the WebApp.

What Are the Characteristics of a Good Information Architecture?

Hardman, Bulterman, and van Rossum [Har93] give a list of ideal requirements for a hypermedia information model—most of which apply equally well to WebApp IAs. They list:

- **Composition with multiple, dynamic data.** The model must support the ability to group different information items into a presentation and the expression of constraints among these items.

  Example: **SafeHomeAssured.com** could include multimedia presentations on the use of the various sensors. Should these presentations occur in a new window? Should they auto-repeat on completion?