INTERACTION DESIGN PATTERNS: P29

Jenifer Tidwell
jtidwell@alum.mit.edu

Abstract:

The fifty patterns contained in this language address the general problem of how to design a complex and interactive artifact. The language's goal is to support high-quality interaction between that artifact and its users. The patterns range from large-scale "genre" patterns, such as Form and Composed Command, to smaller special-purpose patterns like Reality Check and Short Description. The language should be of particular interest to software designers, although the patterns seek to capture design patterns across many different media, including print and industrial design. The language does not address implementation.

Author's Note:

This paper is an excerpt from a much longer work. The patterns chosen for this paper form a loosely-defined "sublanguage" that supports the Navigable Spaces pattern; they frequently refer to each other, but they also refer to other patterns not contained in this paper. To put the patterns in their proper context, the entire table of contents is reproduced here, with the excerpted parts and patterns listed in boldface.

The complete pattern language can be found in HTML form at this Web site:
`http://www.mit.edu/~jtidwell/interaction_patterns.html`

Table of Contents:

Introduction

The patterns contained in this work address the general problem of how to design an artifact, particularly a complex and interactive one. They are intended to be used by people who design user interfaces, Web sites, books, control panels, and other such things. Others who may be interested include people who implement such artifacts, or test them for usability, or manage teams who design and implement them; the language does not attempt to address implementation issues, however.

These patterns are intended to form an Alexandrian pattern language, as found in Christopher Alexander's book *A Pattern Language*, and not a catalog such as is found in the book *Design Patterns: Elements of Reusable Object-oriented Software*. This means in part that they are intended to be used together synergistically, in a way such that the whole is more than the sum of its parts. Like other such pattern languages, it does not break new theoretical ground or present innovative new techniques—more than likely, you have already seen examples of every pattern in here. Instead, it captures ordinary design wisdom in a practical and learnable way.

The intended goal of the pattern language is deliberately broad: to support high-quality interaction between a person and a created artifact. In this context, "artifact" refers to any human-designed thing, real or virtual, which supports one or more of a wide spectrum of activities, ranging from the most passive—absorbing information with little or no interactivity—to the hands-on creation of other objects. The medium could be paper, as with books, forms, or charts; it could be hardware, as with consumer electronics or control panels; or it could be virtual, as with video games, desktop GUIs, palmtops, the Web, or virtual reality.

Naturally, this is an incredibly broad set of artifacts. They are infinitely diverse in their specific goals, their degrees of interactivity, their balance between the verbal and the non-verbal, and other factors. But the best ones all do one or both of these things well:

1. They shape the user's understanding of a set of facts or ideas through a stylized presentation that unfolds them to the user in an appropriate way. A successful artifact will enable its users to completely understand the content being presented.

2. They enable a user to accomplish a task by progressively unfolding the action possibilities to the user as they work with the artifact. A successful artifact will "flow" so well that it lets its users focus entirely upon the task at hand, causing the artifact itself to fade from the user's awareness.

These twin goals, along with corollary issues such as enjoyability, user empowerment, and learnability, define "high-quality interaction" for the purposes of this pattern language. Some artifacts concentrate more upon one aspect than the other, depending upon their context of use. The two aspects are almost orthogonal, but not quite: something which chiefly provides a set of actions still has to present those actions in a comprehensible way, and something which chiefly presents a set of facts or ideas may have to provide the user with actions they need to interact with the artifact.

In both cases, there is an "unfolding" process going on between the artifact and the user. In something which presents facts or ideas (such as a map), that unfolding may be top-down, in which the "big picture" is shown first and the user works their way down into the details as they need. Or, it might be in the form of a fictional narrative, in which the author uses language and character to let the central themes slowly unfold. The basic shape of the content might take the form of one of these "primary patterns," or genres:

- **Narrative.** Examples include works of fiction, nonfiction books, and news articles written in the traditional pyramid form. These are usually linear, and usually verbal, but the unfolding process may work in different ways, as described above.

- **High-Density Information Display.** Maps, tables, and charts fall under this category. Users may approach one of these with the intent of getting a big picture, or of finding specific details; the patterns chosen to build it should support both.

- **Status Display.** A wall clock, a car's dashboard, and a VCR display are all forms of a status display; they are used to monitor the state of something that will change, and like High-Density Information Display, the user must be able to find specific information quickly as well as get a big picture.

In an artifact which presents actions, such as a GUI "wizard," the user may first be presented with a small range of available actions, one of which is taken; then a new set of possible actions is shown, and the user takes one of those; and so on. Alternatively, the range of actions might be very wide, as with some direct-manipulation interfaces. Ultimately, the artifact lets the user accomplish some principal task, which may require smaller supporting tasks to be accomplished first, which may in turn require still smaller supporting tasks. The sense of "flow" comes from being able to do these with an appropriately small amount of time and effort, so that the user never loses focus on their principal task (see Laurel, *Computers as Theatre*).

The primary patterns for actions—their basic shapes—might include these familiar genres:

- **Form.** Tax returns and catalog orders are typical examples, and interactive forms are very common in computer interfaces. The user is expected to provide preformatted information, often in a linear fashion. The available actions are fairly narrow, prescribed by cultural expectations.

- **Control Panel.** A lightswitch is a very simple example, offering an extremely limited choice of actions; a TV remote control is more complex; a nuclear power plant's control room may reach or exceed the limit of a human being's ability to comprehend complexity. Control Panels are used to set the state of one or more things. Cultural constraints often help define the available actions, but as illustrated by the examples, there is a wide range of action availability, so the unfolding process can take place in many ways.

- **WYSIWYG Editor.** A typewriter or word processor, a drawing program, and a forms designer such as Visual Basic are all typical examples. These offer many possible actions to the user at once, including various kinds of direct manipulation, and the results of those actions are usually immediate—they thus support quick, iterative, creative work that often leads to the aforementioned sense of "flow."

- **Composed Command.** Command-line-driven software of all kinds, such as the UNIX operating system or software debuggers, fall into this category; so does one person telling another person what to do, or a person verbally addressing a computer (as in Star Trek). The available actions may be extremely broad, especially if natural language is fully used. This pattern is linguistic where WYSIWYG Editor is graphic: both are highly interactive and user-driven, both offer broad ranges of actions with complex unfolding techniques, and both usually provide immediate feedback. These all work to support complex tasks.

- **Social Space.** This includes the on-line equivalents of "real" social spaces—newsgroups, email lists, chat rooms, and so on. This pattern is unusual in that the artifact is merely a mediator between people, not a direct participant in a dialogue with the user, nor a passive provider of content. Nevertheless, it still reveals content (the conversations taking place) and provides actions (what the user can do in that space), but in a very stylized manner.

These primary patterns form the backbone of this pattern language. They set the tone for an artifact—when a user identifies an unfamiliar artifact as belonging to one of these (or others not explicitly defined in this language, such as a spreadsheet), they tend to make some initial assumptions about its behavior, based on cultural expectations of how these genres usually work. ("What does it do? How am I expected to interact with it? What do I do first? How will it react?") At the end of this introduction, there is a description of each genre's "sublanguage," or a set of interrelated patterns that often work well to support that genre.

Note that the patterns aren't meant to be straitjackets—for instance, they can be used in combination with each other, such as putting bits of Narrative into the cells of a Tabular Set, or using a Form as an adjunct to a WYSIWYG Editor—but the genre boundaries are basically respected by mainstream artifacts, and for good reason. Usability is improved when users' expectations, subconscious or explicit, are followed. On the other hand, experiments, cutting-edge interfaces, and art often make it a point to violate those boundaries. It all depends upon your purpose.

The next set of patterns capture ways of unfolding an artifact's content or available actions. Some apply to content, some apply to actions; but there's no reason to be dogmatic about their use.

- **Navigable Spaces** is highly interactive, letting a user move through the artifact at their own pace, and in their own directions.
- **Step-by-Step Instructions** prescribes a sharply limited set of actions to the user, usually in a linear fashion, moving progressively through stages.
- **Small Groups of Related Things** loosely organizes visual content into distinct groups, usually done hierarchically, so that a user can see both the "big picture" and fine detail.
- **Hierarchical Set** is a way of implementing a High-Density Information Display with a strict tree-like organization.
- **Tabular Set** also implements that pattern, but as a table; again, users can see both the big picture and fine detail.
- **Chart or Graph** is yet a third implementation of High-Density Information Display, but a more graphic one than the other two.
- **Optional Detail On Demand** lets a user reveal hidden actions or content at their own discretion.
- **Disabled Irrelevant Things** blocks certain actions or content according to the current state of the artifact, usually driven by user interaction.
- **Pointer Shows Affordance** temporarily reveals possible actions according to the user's explicit focus of attention.
- **Short Description** temporarily reveals content according to the user's explicit focus of attention.

Other categories of patterns describe an artifact's use of space and other resources, navigational techniques, different kinds of actions, the interrelationships between an artifact's "working surfaces," and so on.

There's one thing you should keep in mind about this language, however, that is atypical of other Alexandrian pattern languages. Most of these patterns can be used at many different levels of scale. A Form, for example, may be the dominant pattern in one artifact, while being a minor helper task in another. Likewise for a High-Density Information Display such as a chart or a table. Small Groups of Related Things is recursive by definition, much like Composite in *Design Patterns*, and the concept of a working surface is also recursive—any single surface can be composed of a set of Tiled Working Surfaces, for instance.

Because of this scale issue, I haven't yet been able to draw a coherent diagram of the whole language, nor define clear linear paths through it. I am open to suggestions on how best to do this. For instance, the sublanguages are not much more than suggestions, based on which patterns seem to go well with each other; it shouldn't be interpreted as exclusive or proscriptive.

How to Use This Pattern Language

At one level, this language is a way to describe existing artifacts. Using it as such should be fairly simple: read through the language, and pick out the patterns that you see. The problem statements and forces in the pattern descriptions may help you understand how the artifact does what it does, and what tradeoffs its designer may have been considering.

At a much deeper level, you can use this language as a tool to help you design an artifact. Please understand that it is no substitute for creativity or a good process. It is not a silver bullet. To use any pattern language effectively, you must start with an understanding of the artifact's purpose and audience, as is the case in any design methodology. (How can you pick a design solution if you don't know what the relevant factors are?) And to get the most benefit out of it, I believe you must allow the design to progress iteratively. Allow it to grow organically, by weeding out the bad ideas as you go and letting the good ones flourish. Start with the broad strokes and work down into the fine details; with each iteration, check your designs against reality, so that you can discover the bad choices and get rid of them.

(This reality may take the form of the viewpoints of different stakeholders, such as users, technical writers, testers, customer educators, and marketers. If they "speak" the same pattern language you do, so much the better—let them in on the fun. They might have some excellent design ideas. In my experience, they almost always do.)

It is common in software to start with a "conceptual model," or a model of objects, relationships, behaviors, and states that describes the artifact independently of the user interface. Upon first reading, these patterns may seem to describe just the surface of an artifact—the way it looks and the way it behaves. But the patterns may also be used to help design the conceptual model behind the interface. For example, content presented as Navigable Spaces might be implemented with one "object" per space, with object relationships corresponding to the links between the spaces. A Hierarchical Set should be the visible manifestation of a tree structure. A Form may present a series of editors for each property of an object, maybe with subforms for subobjects.

Some designers start from the user's perspective. They first design the interface, based on the user's needs and goals, and then design the underlying model to match it. This works well if you have the luxury of being able to affect the design the whole artifact. Conversely, you could start with the conceptual model—perhaps it is an unchangeable requirement—and then choose interaction patterns that have high fidelity to that model, assuming the model is a good one to start with.

The point is, users are going to build their own mental models about the artifact from what they can see. If there is consistency between the underlying model and the interface, and the interface is designed well enough to convey that model effectively to the intended audience, then the users will build mental models which correspond pretty well to the artifact's underlying model. Then the artifact has integrity. The user can more easily predict the artifact's behavior. Errors, if they happen at all, become comprehensible. The interface is easier for the designer to maintain over time, as the model changes.

In any case, I am not going to present a failsafe method for using this pattern language. In the first place, it has not been used enough to generalize from successful examples. Secondly, and perhaps more importantly, I'm not convinced that it will fundamentally change the processes that designers already use to design artifacts. A pattern language should make the processes smoother, more effective, faster, and hopefully with better end results; but there is already a wealth of knowledge out there on how to do design. As pointed out above, user contact and iteration are the keystones of good design.

Acknowledgements

- John Vlissides, for shepherding this pattern language for PLoP'98

- Jim Coplien, for excellent feedback and general advice
- Tom Erickson, for early reviews and support
- The Boston Patterns Group, for a writer’s workshop on some of these patterns
- The “Gang of Threes”—David Cymbala, Austin David, and Mike Wilson
- Dan Palanza, Brad Appleton, Mike Anderson, Bob Chassell, and others that sent in comments
- My managers and colleagues at InConcert, Inc.

Navigable Spaces

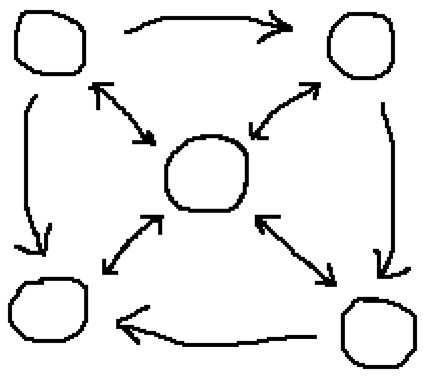
Examples:
- The WWW and other hypertext systems
- Myst
- Museum exhibit in a set of physical rooms
- Set of applications in a suite, as with the PalmPilot or a network computer

Context: The artifact can be organized into distinct conceptual spaces or working surfaces which are semantically linked to each other, so that it is natural and meaningful to go from one to another.

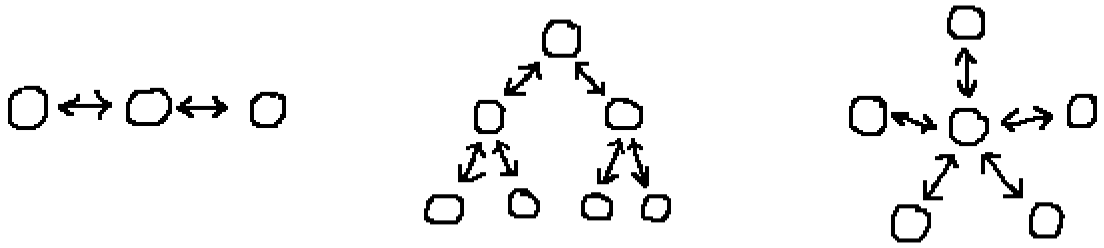
Problem: You have a large amount of content—too much to be reasonably presented in a single working surface. How can you organize the content so that a user can explore it at their own pace, in a way which is comprehensible and engaging to the user?

Forces:
- The user wants to know where they can (or should) go next, and how it's related to where they are now.
- The user wants to be able to choose where to go next.
- The user doesn't want to get lost.
- The concept of information spaces is a natural one for people to think about, both because it mirrors the real world and because the WWW is so commonly understood.
- It's delightful to explore a new place, where the user doesn't necessarily know what's "around the corner."

Solution: Create the illusion that the working surfaces are spaces, or places the user can "go" into and out of. Start out with one top-level or "home" space, to which the user can easily return. In each space, clearly indicate how you get to the next space(s), such as by underlined text, buttons, images of doors, architectural features, etc. Use the spatial locations of these links to help the user remember where the links are. Provide a map of how the spaces are interconnected (*Map of Navigable Spaces*), preferably one that allows the user to go directly to the spaces represented on the map. Make sure that the user can easily retreat out of a space (*Go Back One Step*) or return to the home space (*Go Back to a Safe Place*).



The user will build a mental model of the content from the structure of the Navigable Spaces. Therefore, construct the spaces and their interconnections to mirror the model you want to present (which may not be the same as the actual underlying data structure). Chains, trees, and star patterns are common ways to structure Navigable Spaces (see illustration below); they are easy to understand, visualize, and navigate, and they can contain rich content.



Resulting Context: As pointed out above, *Map of Navigable Spaces* should be one of the first patterns you deal with, even if you explicitly choose not to use one; the same for *Go Back One Step* and *Go Back to a Safe Place*. To help show where the links are in the spaces, you can use *Pointer Shows Affordance;* to give additional information about where they go, use *Short Description.*

People using the WWW tend to depend upon their browser's *Interaction History* (the links you've most recently visited, in chronological order) to get around. Not surprisingly, they also depend upon their *Bookmarks* to keep track of places they want to go back to. These two patterns might be especially important in any large or unbounded set of Navigable Spaces, particularly if a map is impractical.

When you're dealing with power users, seriously consider the value of displaying more than one surface at a time, perhaps using *Tiled Working Surfaces*. It's often good to provide the user with the option of being in at least two or three spaces of their choice, especially if a user is likely to be jumping between spaces frequently. This does increase the user's cognitive load, though, so it may not be appropriate for simpler artifacts that require short learning curves.

Notes: With games, part of the fun is in figuring out where you are and where you can go next, so maps and obvious links would actually reduce the user's fun. In a way, the WWW is similar—who could ever make a map of the WWW anyway?—but, of course, not everyone uses it for fun.

Notice that chains are structured similarly to *Step-by-Step Instructions*, trees to *Hierarchical Set*, and stars to *Central Working Surface*. All three of these archetypes have very strong, simple geometric properties; they probably warrant further exploration.

Map of Navigable Spaces

Examples:

- Book's table of contents
- Web site map
- Geographic map
- Selection area for a suite of applications, as with the PalmPilot or a network computer
- Macintosh desktop
- Windows Explorer

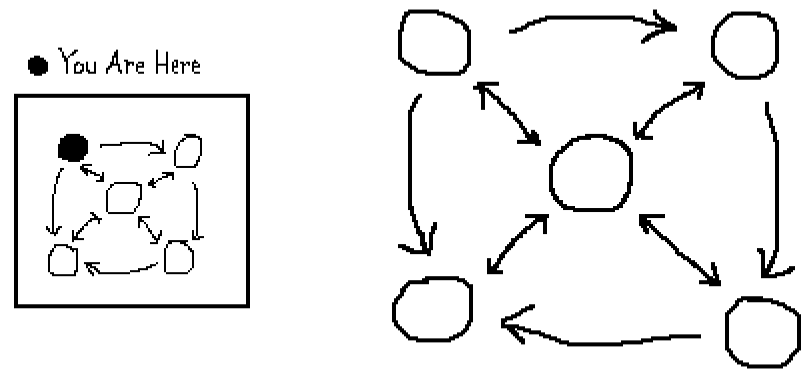
Context: The artifact (or its content) can be organized into distinct spaces or working surfaces which are semantically linked to each other, so that it is natural to go from one to another.

Problem: How can the artifact help a user navigate effectively and remain oriented?

Forces:

- The user should know where they are at any time, in a web of *Navigable Spaces.*
- The user wants to know where they can go next, and how to get there.
- The user should be able to see the interrelationships between the *Navigable Spaces* (or objects, as the case may be); this informs them about the artifact's overall structure.
- The user may need a way to jump from place A to place B, but A may not have a direct link to B—it may be in a totally different part of the artifact.

Solution: Provide a map or diagram of the *Navigable Spaces* relevant to the artifact. Organize it appropriately, and put it where the user can easily get at it; if possible, let it be seen side-by-side with what the user is doing. At all times, show the user where he or she currently is (and there may be multiple places). If possible, allow the user to jump from one place to another by manipulating or gesturing to the map.



Resulting Context: The correct organization of the Map of Navigable Spaces is extremely important. If the spaces are related via a hierarchy, use a *Hierarchical Set;* if they form a flat set, show them in a linear fashion, as with *Choice from a Small Set;* if they form a complicated graph, draw the graph. *High-Density Information Display* may give you some ideas.

You also need to figure out how to represent the sites on that map. Good text labels are important, of course; make them accurate, concise, and descriptive, so users can easily find what they need. Pictorial labels are visually interesting, but run the risk of being cryptic; still, they have been used to good effect in Web site maps that show the sizes and types of the pages' content. Try combining them with text labels.

Some sets of spaces, such as Web sites, are so large that you don't want to show them all at once. Interactive hierarchies may let their nodes be opened and closed; this is one way to cope. *Optional Detail On Demand* may also help. In any case, use your judgment and user testing to find the right amount of material to show.

Notes: For Web sites and other computer interfaces, if you use the common idiom of putting the map on the left side of the screen and a view onto the chosen "space" on the right, you've effectively created a *Stack of Working Surfaces* out of this pattern! It works well, and many people are familiar with it by now.

Go Back One Step

Examples:

- The “Back” button on a Web browser
- The “Back” button on a wizard
- Turning back a page in a physical book or magazine
- The “Undo” feature on some computer applications

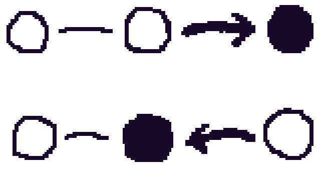
Context: The artifact allows a user to move through spaces (as in *Navigable Spaces*), or steps (as in *Step-by-Step Instructions*), or a linear *Narrative*, or discrete states.

Problem: How can the artifact make navigation easy, convenient, and psychologically safe for the user?

Forces:

- Users tend to explore a navigable artifact in a tree-like fashion, going down paths that look interesting, then back up out of them, then down another path.
- The user may want to temporarily look back at the previous space or state they were in.
- If the user gets into a space or a state that they don’t want to be in, they will want to get out of it in a safe and predictable way.
- The user is more likely to explore an artifact if they are assured that they can easily get out of an undesired state or space; that assurance engenders a feeling of security.

Solution: Provide a way to step backwards to the previous space or state. If possible, let the user step backwards multiple times in a row, thus allowing them to backtrack as far as they want.



Resulting Context: Having a “back” function implies having a “forward” function; this is more of a convenience than a distinct pattern, but Web browsers have set up this expectation, so your users may be unpleasantly surprised if it’s not there. Also, *Go Back to a Safe Place* is a logical pattern to use in addition to this one.

If the user knows they can step backwards multiple times, they may then expect that they can see the history through which they are backtracking—in other words, their *Interaction History*. But don’t discard the history information as the user backtracks through it; the user is probably still interested in it! (The Netscape Web browser behaves this way if you go back a few steps and proceed down another path.)

Notes: A 1994 paper on the usage of Web browsers discovered that on average, the use of the “Back” button accounted for 40% of a user’s actions. This was second only to following an actual link, which made up 52%. (In contrast, the “Forward” button only accounted for 2%.) Reference: “Characterizing Browsing Strategies in the World-Wide Web,” by Lara D. Catledge and James E. Pitkow.

Interaction History

Examples:

- The “history” or “visited links” feature on a Web browser
- UNIX shell’s saved command history
- Logs of exchanged email or other social exchanges

Context: The user either performs a sequence of actions with the artifact, or navigates through it. This may happen in *Navigable Spaces, Control Panel, WYSIWYG Editor, Composed Command*, and *Social Space*.

Problem: Should the artifact keep track of what the user does with it? If so, how?

Forces:

- People are forgetful of tedious details; users are not likely to remember just what they’ve recently done with the artifact, and computers are better at it than people are.
- The user may need to know exactly what they’ve just done, so they can undo their work or backtrack.
- The user may want a high-level overview of what they’ve done, to gain understanding that they wouldn’t get just from memory.
- Audit logs are sometimes necessary, such as with legal regulatory requirements.
- Highly interactive artifacts may generate huge amounts of recordable detail.
- Describing certain actions in a human-readable way is difficult.

Solution: Record the sequence of interactions as a “history.” Keep track of enough detail to make the actions repeatable, scriptable, or even undoable, if possible. Provide a comprehensible way to display the history to the user; most artifacts that implement this pattern use a textual representation, especially *Composed Command*, but that’s not a requirement. (In fact, a history for *Navigable Spaces* may be better portrayed as a state diagram, showing single steps, backtracks, etc.) If the artifact is capable of saving its state, as with *Remembered State*, give the user the option of saving the history from session to session.

It may not be necessary to record every single transaction. Web browsers keep track of the visited sites, which is what the user presumably wants to know; they don’t record printing, saving, or preference changes. But the user should have some control over how big the history gets. This could take the form of a number of history records to keep, or an expire time, or a decision to discard the entire history at the close of a session.

Resulting Context: Now that the artifact has a mechanism to keep track of the history, the user may expect that those actions are scriptable; consider implementing a *Scripted Action Sequence* based on those mechanisms. When found in *Navigable Spaces*, it also provides raw material for *Bookmarks*, if a user can pore over the history and pick out certain points of interest.

Having a history around provides the user with a set of milestones that they can use with *Go Back to a Safe Place*—but explicitly think about whether you want to actually undo all the history between the “present” and the point in the history that the user wants to fall back to. The answer will depend upon your specific
circumstances.

Notes: Jakob Nielsen pleads for better visualization of Web browsers' navigation histories in his November 1, 1997 *Alertbox* column:

> "Well, we can now sort the history list so that all the pages visited on a given site are listed together, but visualization is still missing. It would be very useful to have active sitemaps that showed the user's movements with footprints, showed additional detail at the current focus of attention while collapsing other regions, and also showed connections to other sites with a preview of the relevant sections of these other sites."

Bookmarks

Examples:

- The “bookmarks” or “hotlist” feature on a Web browser
- Actual bookmarks in a book

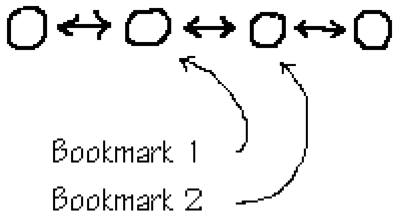
Context: The artifact is large or complex, and allows the user to move freely through it. *Navigable Spaces* is a common pattern to find this in, as is *Narrative.*

Problem: How can the artifact support the user’s need to navigate through it in ways not directly supported by the artifact’s structure?

Forces:

- A user may want to keep track of the places that are most interesting or most useful, for future reference.
- A user may need to temporarily stop using the artifact, but with the intention of coming back later to pick up where they left off.
- A user in “browse mode” may quickly visit a place, decide it’s worth coming back to later, and keep going in a different direction; they need to keep track of where they want to return.
- A user may only be able to be in one place at a time, but want to switch between two or more places at once, perhaps because they want to compare those places.
- Users get a sense of ownership and control over the artifact when they can modify it to suit their needs.

Solution: Let the user make a record of their points of interest, so that they can easily go back to them later. The user should be able to label them however they want, since users are in a better position to choose labels that are memorable to them (see also *User’s Annotations*). Support at least an ordered linear organization, so that a user can rank them according to whatever criteria they choose; if possible, support a grouping structure of some kind. Save the *Bookmarks* for later use.



Resulting Context: You can use *Editable Collection* as a way to let the user modify the set of Bookmarks. If the user can group them (e.g. *Small Groups of Related Things*) or categorize them, consider showing them as a *Hierarchical Set*. *Remembered State* can be used as a mechanism for saving the Bookmarks.

If you provide a sufficiently rich organizing principle, then someone who uses Bookmarks extensively may develop something approaching a customized *Map of Navigable Spaces*. This is good, because that user has now completely adapted their surroundings to their unique way of working. Jakob Nielsen points out in his May 1, 1997 *Alertbox* column that a user may use bookmarks to make their own “map” of the site, and not use the one provided for the site—this may cause trouble for the designers of the site, who can no longer assume that a user has entered a site by one single prescribed path.

Notes: The name of this pattern is stolen shamelessly from Netscape Navigator. It seemed to fit better than
any other name I could think of.

Pointer Shows Affordance

Examples:

- Crosshair or paintbrush pointer in a drawing program
- Arrow pointer over the corner of a resizable window or a movable split pane
- Finger pointer over a hyperlink, especially pictorial links
- Buttons whose borders appear as you move over them

Context: The artifact contains a visual pointer, or "virtual fingertip" (mouse or pen point, for instance) that is the focal point for the user's interaction with the artifact. Patterns that tend to use this a lot are interactive ones with a heavy visual component, including *Navigable Spaces*, *WYSIWYG Editor*, and *Form* (particularly for controls like *Forgiving Text Entry* and *Editable Collection*).

Problem: How can the artifact indicate that a visual entity represents an action that the user may take?

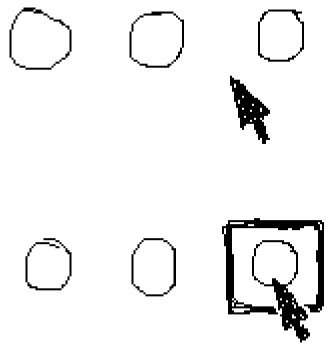
Forces:

- Static visual affordances aren't always enough to indicate the presence of a manipulable control, especially when space is tight or when an elegant-looking graphic design is of paramount importance.
- Multiple affordances can be more effective than one, if they work together properly.
- If the pointer is near the border of a manipulable control, especially something with a small target area, the user should get feedback that instantly tells them whether they're "in" or "out."
- Textual help (e.g., *Short Description*) is not as effective as a simple picture to indicate some kinds of actions.

Solution: Change the affordance of the thing as the pointer moves over it. This can be done in one of two ways: by changing the pointer to a small picture illustrating what can be done, or by changing the thing itself to make it stand out visually.

If you change the pointer, use a small picture illustrating what can be done. Use a standard icon if an appropriate one can be found—crosshairs for drawing, single arrow for selection, I-beam for text entry, hands, pencils, paintbrushes, resize arrows, etc.—because they are so easily recognized. Keep it small or mostly transparent, so that the user can easily see what's under it.

If you change the thing itself, you have a lot of freedom to experiment. Any visual change may be enough to tell a user that the object is at least clickable; but consider your audience when deciding how flashy or distracting the change is. To be sure that your design actually works, of course, you should test it with potential users.



Resulting Context: Be careful not to use this pattern gratuitously. Now that the tools to implement it are widely available, lots of user interfaces use it as a substitute for static visual affordances. This isn't always wise. Think about the poor user looking at a screenful of borderless icons, some of which are buttons, some of which are moveable objects, and some of which don't do anything at all! The user now has to move the pointer over each object in question to see what it does. *Short Description* has the same problems in these cases.

For those of us stuck with non-tactile interfaces, such as mice, this pattern produces something like a substitute tactile sense. As you run the pointer over the interface, you get visual responses that correlate to physical sensations—bumpiness (raised button edges), heat (when something turns from a muted color to a bright color), etc. Says David Cymbala:

> "I was cruising the web the other day, and I was using the mouse pointer to 'brush' across an image map that had patches of 'active' areas.The image jumped into my mind of what I was doing: 'Feeling' the image map with the mouse. Instead of a 'tactile' sensation, I was correlating the image of the mouse pointer with the movement of my hand through space. I almost 'felt' it physically... The pointer allows me to 'feel' visual space as a replacement for the lost tactile dimension." (From personal correspondence, dated June 17, 1998.)

Notes: I find that when I'm working quickly, I depend very heavily on the fact that my pointer changes when I'm over a manipulable control; if I want to resize a window, and I move the pointer towards the window edge, I instinctively start the press-drag motion the instant that pointer changes. I don't actually look hard to see if the pointer is over the control. That zone could extend ten pixels beyond the window edge, for all I care. Conversely, it's very hard to deal with direct manipulation if the cursor doesn't change—I have to pay far too much attention to the screen, and use better fine motion control; and with small controls, there's always this vague uncertainty that the action will succeed.

For some wonderfully bad examples, take a look at the on-line Interface Hall of Shame. Look under the "Visual Elements" section, especially at the Microsoft examples and the first WebZip commentary.

Short Description

Examples:

- Windows tooltips
- Mac bubble-help
- Status bar help
- Captions in drawn figures

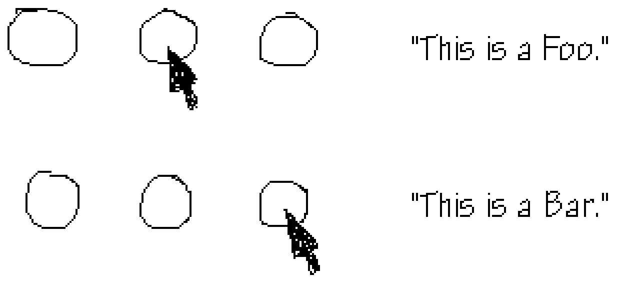
Context: The artifact contains a visual pointer, or "virtual fingertip" (mouse or pen point, for instance) that is the focal point for the user's interaction with the artifact. Nearly all the primary patterns with a visual component can use this to good effect, particularly *Navigable Spaces* for link descriptions, *High-Density Information Display*, *Status Display*, *Control Panel*, and *WYSIWYG Editor*.

Problem: How should the artifact present additional content, in the form of clarifying data or explanations of possible actions, to the users that need it?

Forces:

- A short explanation may be all the user needs or wants; something long will be overkill.
- Users generally don't want to leave the artifact and go somewhere else for help, such as a manual; this usually breaks one's concentration and costs too much time.
- There isn't room to put static descriptive text into the artifact, or visual elegance precludes doing so.
- The descriptive text might be useless most of the time, and may become irritating if it is static or hard to turn off.

Solution: Show a short (one sentence or shorter) description of a thing, in close spatial and/or temporal proximity to the thing itself. Allow the user to turn it on and off, especially if the description obscures other things or is otherwise irritating; alternatively, don't show it without some deliberate user action on an item-by-item basis, such as pressing a key or hovering over the item for a certain length of time.



Resulting Context: You get to decide what text to put into the Short Description. There's no point in being redundant with whatever's statically shown in the artifact; if you're going to impinge upon the user's attention with a popup or something, at least add some value with it. You could use it to describe a possible action (as with *Pointer Shows Affordance*), or describe the results of the action, or reveal more data (thus implementing *Optional Detail On Demand*).

Notes: In his January 11, 1998 *Alertbox* column, Jakob Nielsen strongly recommends using link titles to help give the user a preview of where a Web link goes; they add important contextual information to the sometimes-mysterious HTML links. These are effectively Short Descriptions.

I’ve never seen it done, but this pattern could theoretically be used with speech in a multimodal interface. As you focus your visual attention on some feature, the Short Description for that feature could be spoken aloud to you.

Disabled Irrelevant Things

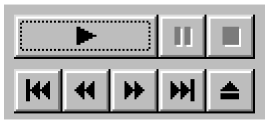
Context: Information or actions that are normally useful become temporarily irrelevant sometimes. This is a common situation in almost all of the primary patterns that use visuals, such as *Control Panel*, *Status Display*, *Form*, and *WYSIWYG Editor*.

Problem: How can the artifact steer the user away from actions that cannot or should not be taken, while still maintaining visual calm and stability?

Forces:

- The artifact should present items in such a way as to allow the user to form a correct mental model of its underlying ideas (for information) or action states (for actions).
- All actions available to the user at a given time should be valid, so that the user doesn't do something erroneous.
- The artifact should be responsible for figuring out what's valid and what's not, to avoid giving the user an unnecessary cognitive burden.
- When the users can trust the artifact to not let them do invalid actions, they will feel more secure about exploring it and trying new things.
- An artifact which is too actively helpful can be distracting, irritating, or confusing.

Solution: Disable the things which have become irrelevant. Hide them entirely if the user shouldn't even be aware of them, or "gray them out" (with their main features barely visible) if the user should know they're there but that they just aren't useful right now. If the thing is a manipulable control, don't allow the user to use it.



Resulting Context: Computer interface toolkits normally provide a reasonable implementation of a disabled or grayed-out state. If the item being disabled uses *Pointer Shows Affordance*, however, remember to disable that too, so that the user doesn't get conflicting cues about whether a given control is usable or not.

As the user uses the artifact, different actions may become available to them as they change the artifact's state over time. This pattern provides one way to let the actions unfold to the user, without the disruption of having items appear out of nowhere. Still, it doesn't really tell the user what to do—it only tells them what they can't do, and it doesn't even tell them why they can't.

Progress Indicator

Examples:

- Countdown timer on a microwave oven
- Progress bars in desktop GUI software
- The percent-complete message in a Web browser during a download

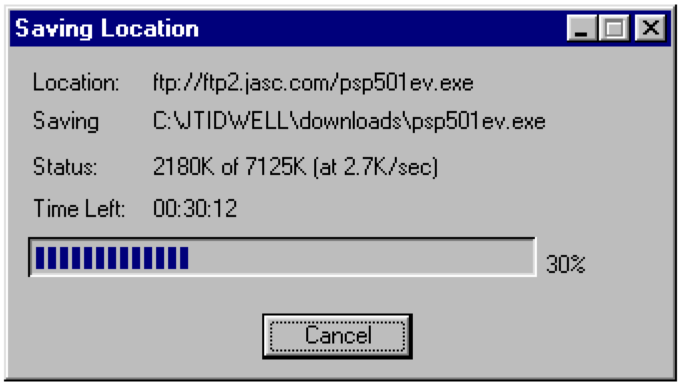
Context: A time-consuming process is going on, the results of which is of interest to the user.

Problem: How can the artifact show its current state to the user, so that the user can best understand what is going on and act on that knowledge?

Forces:

- The user wants to know how long they have to wait for the process to end.
- The user wants to know that progress is actually being made, and that the process hasn't just "hung."
- The user wants to know how fast the progress is being made, especially if the speed varies.
- Sometimes it's impossible for the artifact to tell how long the process is going to take.

Solution: Show the user a status display of some kind, indicating how far along the process is in real time. If the expected end time is known, or some other relevant quantity (such as the size of a file being downloaded), then always show what proportion of the process has been finished so far, so the user can estimate how much time is left. If no quantities are known—just that the process may take a while—then simply show some indicator that it's still going on.



Saving Location

Location: ftp://ftp2.jasc.com/psp501ev.exe
Saving C:\JTIDWELL\downloads\psp501ev.exe
Status: 2180K of 7125K (at 2.7K/sec)
Time Left: 00:30:12

Animation is often used to good effect in this pattern; motion draws the user's attention, and its cessation implies a new relaxed, stable state of being ("the process is done, so you can relax now"). Sound can also be used this way, for the same reason. Be subtle, though, and be aware of the importance of the process relative to the other things demanding the user's attention. For example, *Background Posture* artifacts sometimes shouldn't have attention-demanding Progress Indicators at all. *Helper Posture* artifacts probably should, and so should *Sovereign Posture* artifacts if waiting on the process is suspending all other activity.

Resulting Context: A user may expect to find a way to stop the process somewhere spatially near the Progress Indicator. It's almost as though the Progress Indicator acts as a proxy for the process itself, in the user's mind. Go with it, and put a "stop" action near or on the Progress Indicator if you can.

Notes: A story about the use of sound as a Progress Indicator: I once had a workstation that had a uniquely noisy disk drive, which worked wonderfully (if unexpectedly) as a Progress Indicator for a lot of my software development activities. It wasn't too loud, fortunately, but it did have distinctive sounds for different classes of activities—one for copying a large file, one for compiling, and so on. If I was waiting for a long compile to finish, I could work on other things without having to watch my monitor; the sudden cessation of the sound told me it was done, in a nice subtle way.

(Also, if my workstation hung or started behaving abnormally, the sound it was making often gave me a clue what was wrong. I got very familiar with the sound and timing of an 8-meg core dump. But that has little to do with Progress Indicators...)