A visual vocabulary for describing information architecture and interaction design

version 1.1b (6 March 2002)

Jesse James Garrett
(contact)

Translations of this document are available:

- Chinese (thanks to Arky Tan)
- Japanese (thanks to Atsushi Hasegawa)
- Spanish (thanks to Javier Velasco)
- Italian (thanks to Laura Caprio and Beatrice Ghiglione)
- French (thanks to Francois Lamotte)
- German (thanks to Marcus Brinckhoff)
- Portuguese (thanks to Livia Labate and Laura Cretton Lessa)

Table of Contents

Summary

Diagrams are an essential tool for communicating information architecture and interaction design in Web development teams. This document discusses the considerations in development of such diagrams, outlines a basic symbology for diagramming information architecture and interaction design concepts, and provides guidelines for the use of these elements.

Version history

1.1b (6 Mar 2002)
    Information on built-in support in OmnGraffle 2.0
    New shape library for iGrafx Flowcharter
1.1a (17 Sep 2001)
    New shape libraries for Macromedia FreeHand
    Posted cheat sheet and PDF shape template
1.1 (31 Jan 2001)
    Added the filestack element
    Added the conditional selector element

Modified the arrow element to allow multiple arrowheads
Modified the cluster element's behavior so that it now appears only downstream from a conditional branch or selector
Modified the conditional branch element's behavior to allow for a null result
Numerous improvements to the shape libraries
New shape library for Adobe InDesign

1.0 (17 Oct 2000)
Initial release

Initial considerations

A visual vocabulary is a set of symbols used to describe something (usually a system, structure, or process). The vocabulary described here may be used by an information architect or interaction designer to describe, at a high level, the structure and/or flow of the user experience of a Web site.

These descriptions, or diagrams, are used by five primary audiences:

- **Project sponsors and managers** use them to obtain a general sense of the scope and form of the project.
- **Content producers** use them to derive content requirements.
- **Visual and interface designers** use them to derive a count of how many unique page designs must be produced, and to obtain an initial sense of the navigational and interface requirements for these designs.
- **Technologists** use them to derive functional requirements.
- **Information architects and interaction designers** use them to develop detailed navigational and interface requirements for each page.

Every one of these audiences (with the exception of sponsors) needs a great deal of detail to do their jobs. The trouble is that the detail each audience requires differs vastly from the detail required by others, and the bulk of this detail is irrelevant to the needs of other audiences.

The sensible approach is to limit the detail in the diagram to that which can be usefully applied by all audiences. The diagram thereby serves as a touchstone document for the development of more detailed documents specific to the needs of each audience.

Some other key requirements of a visual vocabulary for information architecture and interaction design include:

- **Whiteboard-compatible:** The vocabulary should be simple enough that diagrams can be sketched quickly by hand. The elements of the vocabulary should be distinct enough from each other that moderately sloppy drawing cannot compromise the clarity of the diagram.

- **Tool-independent:** The vocabulary should be designed so that specialized software tools are not required in order to construct diagrams. The vocabulary should not favor the use of any particular software tool, but should instead enable architects to work with the tools they are most comfortable using.
- **Small and self-contained:** Because the diagrams are used by a diverse range of audiences with differing levels of knowledge of (or even interest in) diagramming systems used in other areas of technical development, the vocabulary should not require such knowledge or interest. The total set of elements should be kept as small as possible, maintaining a strict one-to-one correlation between concepts and symbols, so that the vocabulary can be learned and applied quickly. The concepts expressed by a diagram may be arbitrarily complex; the means of their expression should not be.

Conceptual underpinnings

Information architecture and interaction design are two sides of the same coin. (See "The Elements of User Experience" for definitions of these terms as they are used here.) Diagrams of contemporary sites inevitably involve both sides. But for each, the objectives of the diagram are slightly different.

In both cases, the diagram focuses on what we call **macrostructure**, providing just enough detail to enable team members to get the "big picture". The task of the architect is to determine the appropriate level of detail to meet this objective. The specific page-level detail, or **microstructure**, is detailed in other documents that the architect may not be primarily responsible for developing.

When describing information architecture, the diagram should emphasize conceptual structure and organization of content. Note that conceptual structure is not the same as navigational structure. The objective of the information architecture diagram is not to provide a full-blown navigational specification; this level of detail is best kept in other documents, where it is less likely to confuse and distract.

When describing interaction design, the diagram should emphasize how the user flows through defined tasks, and what the discrete steps are within these tasks. As with navigation, details of interface should not appear in the diagram -- if you find yourself drawing buttons and fields, you're probably loading the diagram down with excess detail.

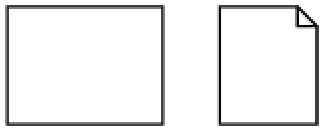
This vocabulary is based on a simple conceptual model encompassing both information architecture and interaction design:

- The system presents the user with **paths**.
- The user moves along these paths through **actions**.
- These actions then cause the system to generate **results**.

Simple elements: pages, files, and stacks thereof

The basic unit of user experience on the Web is, of course, the **page**, which we represent as a simple rectangle. Note that the page is a unit of *presentation*, not (necessarily) a unit of implementation -- one page in your diagram may correspond to multiple HTML files (as in a frameset interface) or multiple units of code (as in a server-side include or database-driven implementation).

In addition to pages, there are also **files**, parcels of data without navigational properties. Files are delivered to the user for use outside a Web browser environment (such as audio or video files, stand-alone documents like PDFs, or executables). For these, we use our old friend the dog-eared document icon.



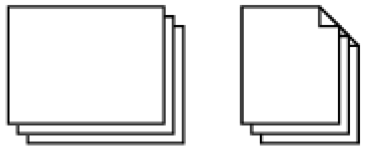


Figure 1a: *[left]* The page and the file
Figure 1b: *[right]* The pagestack and the filestack

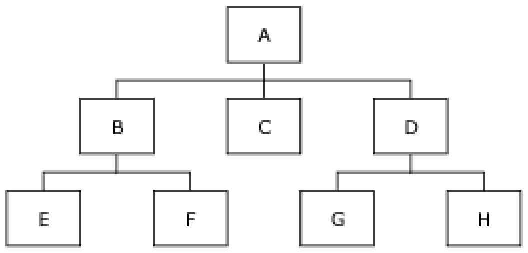
Use a **pagestack** to indicate a group of functionally identical pages whose navigational properties are immaterial to the macrostructure of the site. Similarly, a **filestack** represents a group of files that receive identical navigational treatment and can be classified as a single entity (such as a collection of downloadable games or a library of PDF instruction manuals).

We use **labels** on pages and files to identify them. These don't need to correlate to actual designations such as HTML <TITLE> elements or filenames, but they do need to be unique for each page or file in the diagram. Unique numeric identifiers and type designations also provide a good way to keep track of all the pages and files in your diagram.

Creating relationships: connectors and arrows

Relationships between elements are depicted with simple lines or **connectors**. These conceptual relationships will inevitably translate into navigational relationships -- but not all navigational relationships will appear in the diagram.

In the case of information architecture, these relationships are commonly reflected through a hierarchical organization of pages into trees. However, this is by no means required or even (in some cases) recommended.



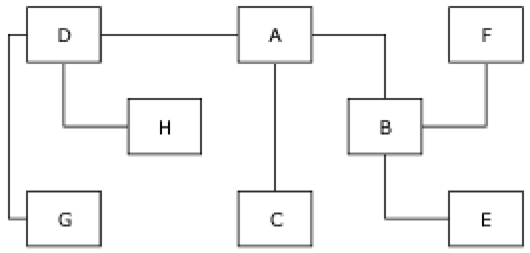
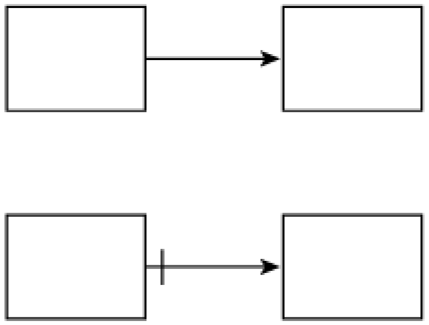


Figure 2a: *[left]* A simple tree structure

Figure 2b: *[right]* The same structure as in 2a, diagrammed differently

When diagramming interaction design, our connectors also need to convey **directionality** to indicate how the user will move through the system toward completion of a particular task. Turning our connectors into **arrows** will do the trick nicely. We use the terms **downstream** and **upstream** to refer to the position of elements relative to this forward movement.

Note that these arrows are not like the arrows indicating a one-way street, but rather like the arrows indicating the way to the food court in the mall. The user is not prohibited from moving in the opposite direction; the arrow merely indicates the direction in which the user is likely to want to go.



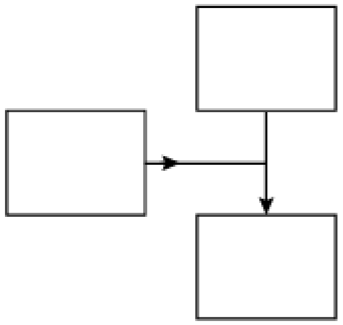


Figure 3a: *[top left]* Arrow indicates downstream movement toward task completion

Figure 3b: *[bottom left]* Crossbar indicates upstream movement is not permitted

Figure 3c: *[right]* Multiple arrowheads clarify directionality

If for some reason we want to prohibit this upstream movement (such as in cases where some irreversible action like deleting a record has taken place), we use a **crossbar** (just a short perpendicular line) on the opposite end of the arrow to indicate this.

In some cases, it may be necessary to place an additional arrowhead near the upstream page to clarify the directionality of the flow in a more complex architecture. (A practical note: Many diagramming applications do not allow the user to string arrows together in this fashion. To work around this, the shape libraries include a "gluedot" element, an invisible element consisting of a single connection point. Use this element to connect arrows together.)

Connectors and arrows can also be labeled, but the use of these should be limited to cases in which the action taken by the user needs to be clarified. If the labels become long and unwieldy and start to clutter the diagram, point the reader toward a footnote or appendix entry.

In the examples given throughout this document, **footnote or appendix references** will appear as a number and letter combination in parentheses. Numbers refer to the diagram page on which the reference appears; letters refer to the specific note. For example, the first note on page 3 of a diagram would be referenced as (3a), the second (3b), and so forth.

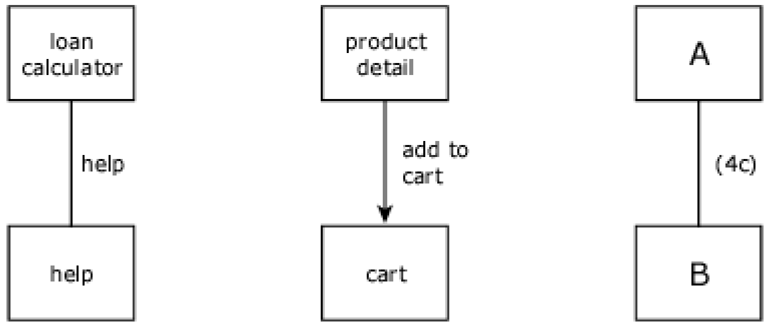


Figure 4a: *[left]* A superfluous label
Figure 4b: *[middle]* A useful label
Figure 4c: *[right]* A footnote or appendix reference

All at once: concurrent sets

A **concurrent set** (represented by the half-circle) is used in cases where a user action generates multiple, simultaneous results (such as spawning a pop-up window at the same time a page is loaded in the main window, or displaying a page while a file is being downloaded).

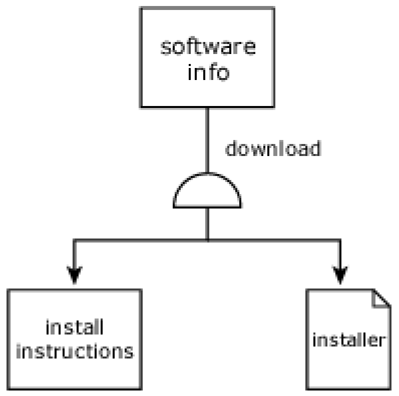


Figure 5: A concurrent set

Like arrows, concurrent sets are directional. Upstream elements connect to the curved side; downstream elements connect to the flat side.

Breaking it up: continuation points

Information architects often find themselves yearning for ever-larger sheets of paper on which to diagram their work. But even if large-format output devices such as plotters were widely available, some architectures are simply too complex to capture in a single, all-encompassing diagram.

To allow us to separate our diagrams into easily digestible sections, we use **continuation points** (square brackets) to bridge the gaps between pages.

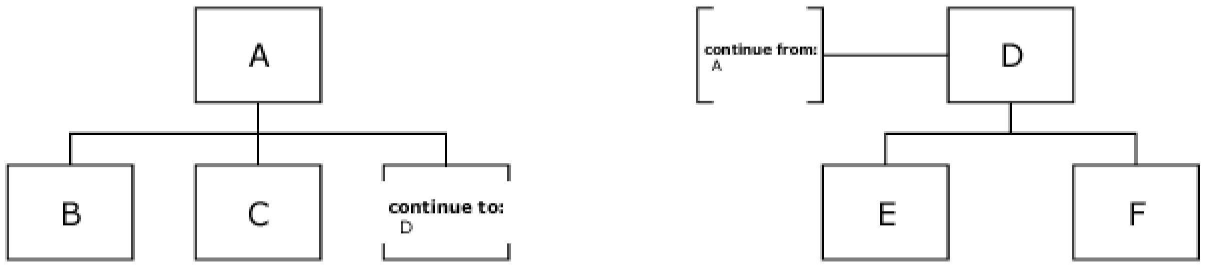


Figure 6a: *[left]* A "continue to" point refers the reader to another diagram
Figure 6b: *[right]* A "continue from" point, picking up where 6a left off

A single continuation point may list one or more sources or destinations as needed. The orientation of the brackets (horizontal or vertical) carries no particular meaning; the choice of orientation is a matter of the architect's aesthetic judgment.

Commonalities: areas and iterative areas

The **area** element (a rounded-corner rectangle) is used to identify a group of pages that share one or more common attributes (such as appearing in a pop-up window, or having some unique design treatment). Use labels to identify these attributes or (as with connectors), refer to notes elsewhere in the document if you have a lot to say.

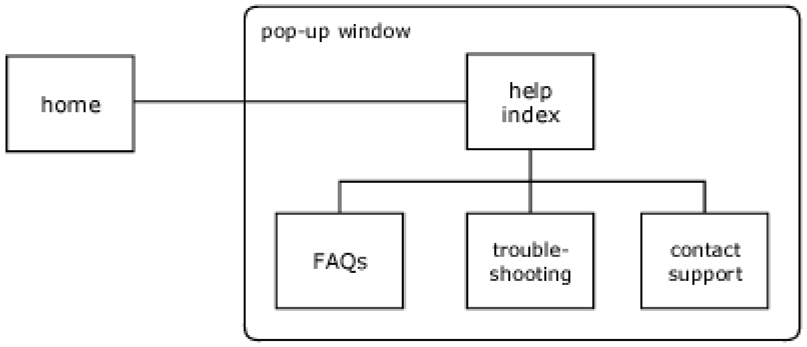


Figure 7: An example use of an area to represent a pop-up window

Many architectures involve repeating the same basic structure as it is applied to a number of functionally identical information elements. For example, you may have a product catalog in which each product has a number of pages associated with it. You could draw an instance of this structure for each product, but why waste your time? Just use an **iterative area** -- a stack of rounded-corner rectangles -- instead.

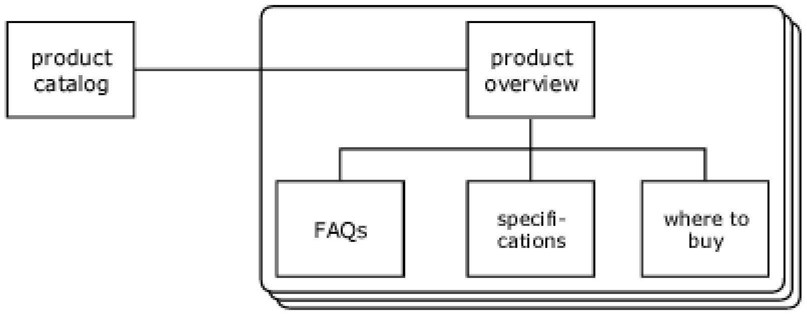


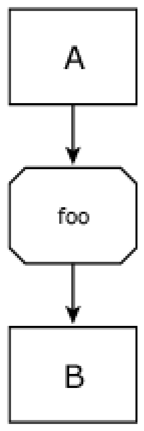
Figure 8: An example use of an iterative area to represent a repeated structure in a product catalog

Note that connectors and arrows don't actually point to the areas themselves. The area elements serve only to enclose the pages. Areas should be applied carefully -- it's very easy to get caught up capturing all kinds of details with area elements that don't manifest in the user experience (such as which pages are hosted on which servers) or otherwise interfere with the diagram's overall objective of communicating the macrostructure.

Reusable components: flow areas and references

Some interaction designs require a sequence of steps (like a login procedure, for instance) to appear repeatedly in different contexts throughout the design. Often these sequences are merely a component of one or more larger tasks the user is trying to accomplish. (This is analogous to the concept of a subroutine in computer programming.)

Such a reusable sequence is called a **flow**, and it is represented in the diagram through two elements: the **flow area**, which encloses the flow itself; and the **flow reference**, which serves as a sort of "placeholder" for the flow in every context in which it is repeated. Both elements have the same basic shape, a rectangle with the corners clipped off (or, if you prefer, a distorted octagon).



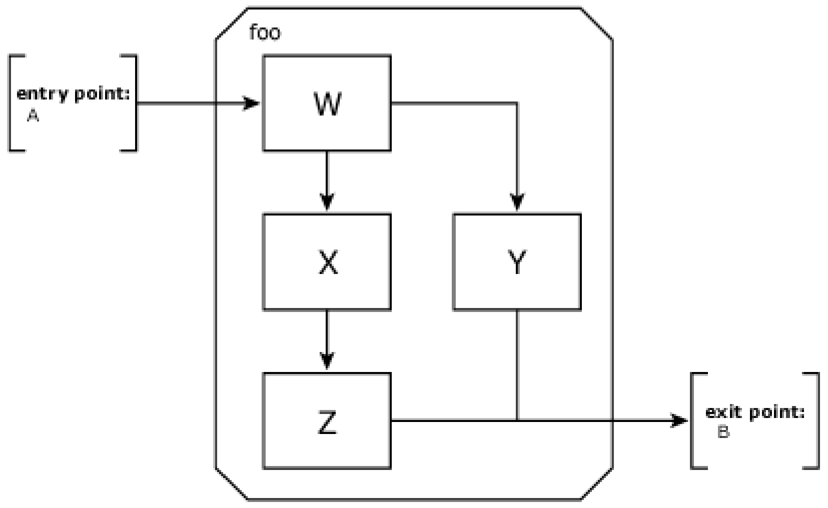


Figure 9a: *[left]* A flow reference serves as both a "continue to" point and a "continue from" point

Figure 9b: *[right]* The flow area referenced in 9a

Flow areas also require the use of two special types of continuation points: **entry points** and **exit points**. These are placed outside the flow area, while continuation points within the flow area indicate that the flow spans multiple diagrams.

Flow references themselves function very much like continuation points. The objective of both types of elements is the same: to allow the architect to break up the diagram across pages. The difference is that a flow reference can serve in both the "continue to" and "continue from" capacities, while a continuation point can only be one or the other. If you don't need an element to play both roles, you probably don't need to use a flow.

Basic concepts for conditional elements

With increasing frequency, information architectures and interaction designs are reshaped dynamically by the system as the user moves through the site. This reshaping is accomplished by means of **conditional logic**, and the remaining elements of this vocabulary are specific to conditional logic structures. Here's a basic conceptual model for application of conditional elements:

- The system keeps track of one or more **attributes**. These attributes may be particular to:
    - the user (such as user type)
    - the session (such as login status)
    - the content being accessed (such as subject matter)
    - or they may exist "in the world" (such as the time or date).
- Attributes have **values** ("3 p.m." is one possible value for "time of day").
- The association of an attribute with a particular value is called a **condition**.
- Conditions are **evaluated** by the system to determine if they are true.

In a static architecture, every path is presented to every user under every circumstance, and each path always leads to the same result. In a dynamic architecture, the system decides which paths or results are presented to the user based upon evaluating one or more conditions.

To minimize clutter in our diagrams, these conditions are typically described in a footnote or appendix entry accompanying the diagram.

Making choices: decision points

When one user action may generate one of a number of results, the system must make a decision about which result is to be presented. (Perhaps the most common example of this is error handling on form submission.) We call this a **decision point**, and as in traditional flow charts, it is represented by a diamond.

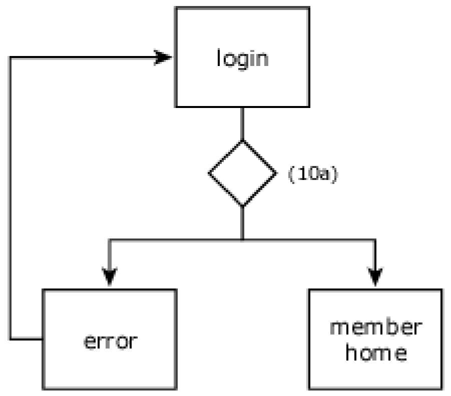


Figure 10: An example use of a decision point in a login sequence

Note that arrows *must be used* in conjunction with decision points to clarify whether associated elements are upstream or downstream from the decision point.

Pathfinding: conditional connectors and arrows

A **conditional connector** (represented by a dashed line) is used when a path may or may not be presented to the user depending upon whether one or more conditions are met.

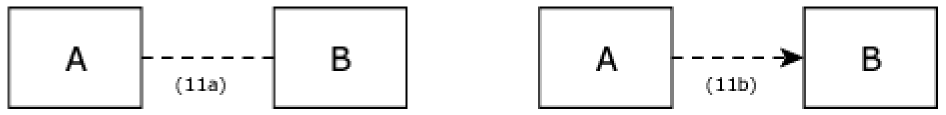


Figure 11a: *[left]* A conditional connector
Figure 11b: *[right]* A conditional arrow

For example, there may be a page containing sensitive information that only company employees should have access to. The condition in this case would be the user type (employee); if the condition is met, the path is made available. If not, no path exists.

Multiple choice: conditional branches

When the system must select one path among a number of mutually exclusive options to be presented to the user, we use a **conditional branch** (triangle). Upstream elements connect to one point of the triangle; downstream elements connect to the opposite side.

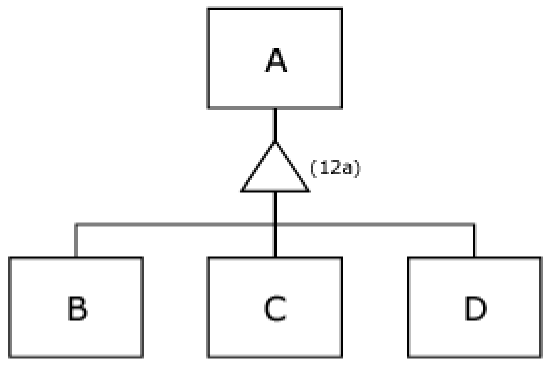


Figure 12: A conditional branch

The example shown in figure 12 looks a lot like the decision point example above in figure 10, but the behavior described here is quite different. In the decision point example, only one path (or navigational element) was presented to the user; where that path took the user was dependent upon certain conditions.

In figure 12, the system is making a similar decision, but it happens *before* the user action. The conditional branch indicates that the system is deciding which path will be presented to the user. The paths from page A to pages B, C, and D are mutually exclusive; for example, if a path to B exists, paths to C and D do not.

As with conditional connectors and arrows, a conditional branch may provide the user with no path at all (a **null result**). The difference here is that with a conditional branch a null result may not be permitted at all; and if it is permitted, it is one of three or more possible results. Indicate whether the branch permits a null result in your footnote or appendix entry.

Choose one or more: conditional selectors

The **conditional selector** element (represented by a trapezoid) functions much like the conditional branch, with one important difference: with the selector, the various downstream paths are *not* mutually exclusive. any number of the paths that fulfill the condition(s) may be presented to the user.

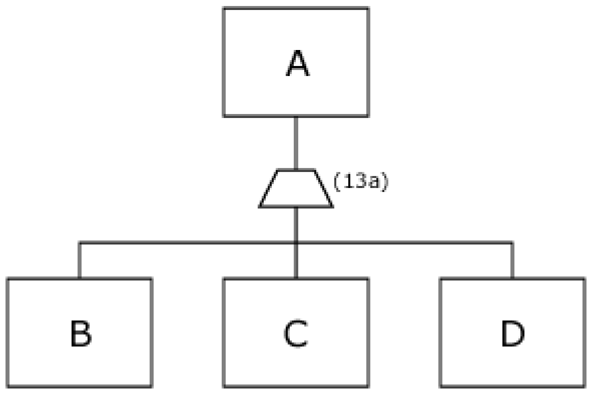


Figure 13: A conditional selector

The most common application of the conditional selector is in results generated by a search engine. In this case, the search results page would appear upstream from the selector; the condition is the search criteria input by the user; the downstream paths would lead to the content pages indexed by the search engine. As with a conditional branch, a conditional selector may generate a null result -- in fact, this is far more common with a selector than with a branch.

One decision, many paths: clusters

Some conditional structures require that the system present more than one path based upon certain conditions. We associate these paths together in the structure with a **cluster** (represented by a circle). The cluster can appear downstream from either a conditional branch or a conditional selector.

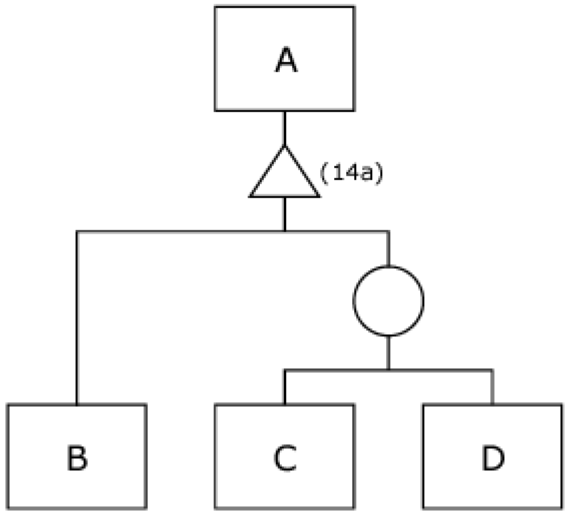


Figure 14: A cluster downstream from a branch

The structure illustrated in Figure 14 functions pretty much like a normal conditional branch, but for one condition we are presenting more than one path to the user. So if the attribute being evaluated has value *x*, the user sees a path to page B; but if the attribute has value *y*, the user sees paths to both page C *and* page D.

Some restrictions may apply: conditional areas

When one or more conditions applies to a group of pages, those pages are enclosed within a **conditional area** -- a rounded-corner rectangle like a standard area, but with the dashed-line treatment of a conditional connector.

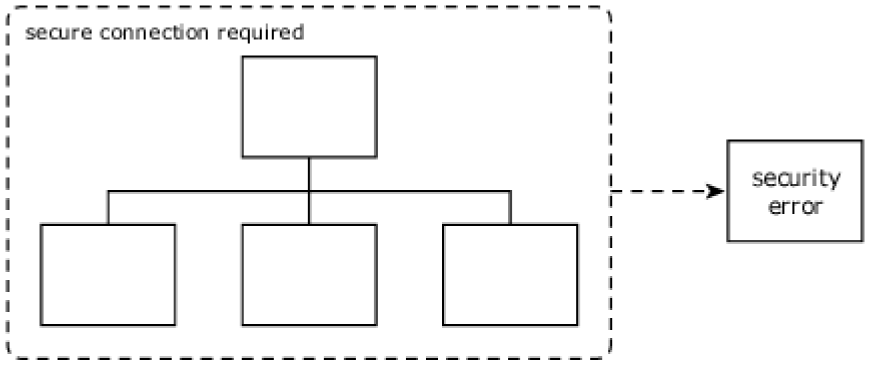


Figure 15: An example use of a conditional area where a secure connection is required

Conditional areas are applied most commonly in situations involving access permissions, such as when a valid login or encrypted (SSL) connection is required. Unlike the other types of areas, conditional areas are associated with a result, which is generated in the event that the condition(s) are not fulfilled.

Conclusion

If you'd like to see how the whole system comes together, here's a sample diagram of the information architecture and interaction design of MetaFilter. (I wasn't involved in the development of this site; this diagram was simply reverse-engineered from it.)

Eric Turner's quick reference guide is an easy one-page description of all the elements. And Scott Larson created this handy cheatsheet for quick reference to the various conditional elements. And for those interested in creating their own shape libraries for use with an application other than the ones below, here's a PDF of all the shapes (thanks to Ross Olson for the suggestion).

This vocabulary necessarily represents only a first step. As information architecture and interaction design for the Web continue to evolve, situations will inevitably arise that this vocabulary does not address. Your feedback and recommendations for the next revision of this vocabulary are welcome.

Downloadable shape libraries

OmniGraffle for Mac OS X is the first application to ship with built-in support for the visual vocabulary. OmniGraffle is currently pre-installed on all Apple Power Macs and PowerBooks. It can also be downloaded from the Omni site.

PocketDraw 2.0 for the PocketPC includes a built-in visual vocabulary palette for the information architect on the go. You can see examples at the PocketDraw site or on Mike Lee's weblog.

Other shape libraries available:

- Stencil file for Visio 2000
- Stencil file for Visio 5
- Stencil file for Visio 4
- PowerPoint file (thanks Fredrik Johansson Oviedo for some good suggestions)
- Library file for Adobe InDesign 2.x (thanks Andrew Robinson)
- Library file for Adobe InDesign 1.x
- Library file for FreeHand 10 (thanks Andrew Crow)

- Library file for FreeHand 9 (thanks Andrew Crow)
- Illustrator EPS file
- Library file for iGrafx Flowcharter 2000 (thanks Andrew Robinson)
- Library file for OpenOffice (thanks Nelson Rodriguez-Peña)
- Collection of EPS files containing one element per file, suitable for import into other applications (1.1 MB)
- Two different SVG libraries: [1] (1.9 MB) [2] (52 KB; thanks to Marcus Brinkhoff)
