Constantine & Lockwood, Ltd.

Working Paper

From Abstraction to Realization in User Interface Designs: Abstract Prototypes Based on Canonical Abstract Components

Larry Constantine,* Helmut Windl,† James Noble,‡ & Lucy Lockwood¶

In July 2000, a group of colleagues convened for a colloquy on the state and future of usage-centered design. This meeting served not only as a forum for review and consolidation of accumulated experience in usage-centered design, but also as a workshop for refinement and improvement of the process, especially with regards to convergence with other design and development processes and models. Out of the discussions emerged a number of conceptual and practical breakthroughs, among them a dramatically improved form of abstract prototype that simplifies and speeds the process of producing high quality user interface designs based on task models.

Learn more about usage-centered design, including training in canonical abstract prototypes, at http://www.forUse.com.

Abstract Prototypes

One of the truly powerful tools in usage-centered design is the abstract prototype [Constantine, 1998; Constantine & Lockwood, 1999]. An abstract prototype allows designers to describe the contents and overall organization of a user interface without specifying its detailed appearance or behavior; it is, thus, a model of the architecture of the user interface being designed. We and our clients have found that abstract prototypes provide an effective bridge between task models based on task cases (essential use cases)[1] and a final design in the form of a realistic prototype, whether on paper or in software. In particular, by maintaining a focus on content, organization, and function independent of layout, appearance, and behavior, abstract prototypes have repeatedly been found to encourage both sound architecture and creative innovation [Constantine, 1998]. Driven by an appropriate task model, abstract prototypes help designers to devise user interface solutions that are both practical and novel [Constantine, 2000].

* Director of Research, Constantine & Lockwood, Ltd., and Professor of Computing Science, University of Technology, Sydney (Australia)
† Director, Usability Competence Center, Siemens AG
‡ Lecturer, University of Victoria, Wellington (New Zealand) and Consulting Associate, Constantine & Lockwood, Ltd.
¶ President, Constantine & Lockwood, Ltd.

The authors may be contacted at Constantine & Lockwood, Ltd., 58 Kathleen Circle, Rowley, MA 01969; tel: 1 (978) 948 5012; fax: 1 (978) 948 5036; mailto:larry@foruse.com
© 2000, Constantine & Lockwood, Ltd.

State of the Abstract Art

In its most common form in usage-centered design, an abstract prototype consists of a content model and a navigation map. The content model comprises a series of views (interaction contexts)[1] populated with abstract components, that is, with the tools and materials needed for users to perform the tasks being supported within each view. The navigation map complements the content model showing the possible paths or transitions interconnecting all the views (interaction contexts) in the user interface.

In a conventional content model, each view is usually represented by a separate, labeled piece of paper onto which abstract components are posted, typically in the form of sticky-notes. Where the distinction makes sense, simple glyphs (small icons) are employed to distinguish materials, which represent the containers and information of interest to users, from the tools that operate on these materials or perform other actions for users. An example of a conventional abstract prototype is shown in Figure 1.

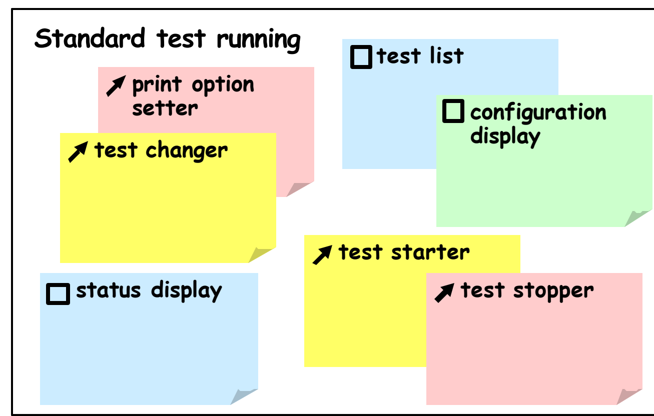

Figure 1 – Example of conventional (fully) abstract prototype.

Other variations of abstract prototypes include “wire-frame” mockups and abstract layout diagrams. Wire-frame mockups, such as the one shown in Figure 2, represent the relative size and position of visual user interface elements. Color-coding of the areas may also be used to indicate the type of element represented or the relative importance or priority of the information or function. This latter variation has enjoyed some popularity among graphic designers for Web-based applications.

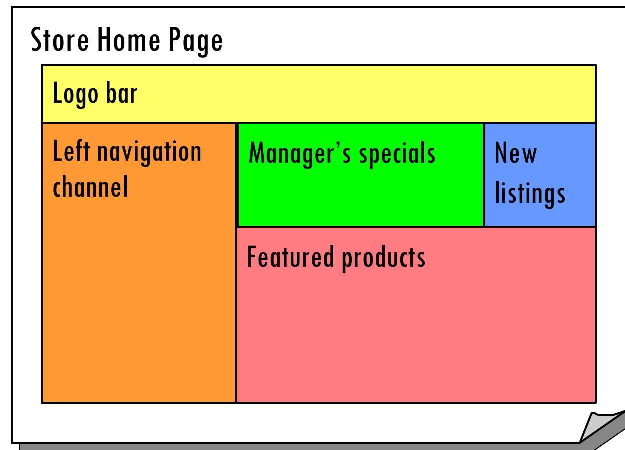

Figure 2 – Example of wire-frame mockup.

An abstract layout diagram, such as the one shown in Figure 3, is a form of "low-fidelity" prototype. It shows the relative size and position of user interface elements, but not their exact appearance.

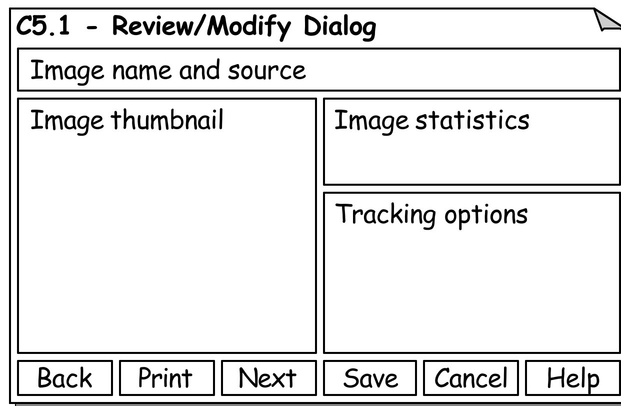

Figure 3 – Example of abstract layout diagram.

The sundry forms of paper (diagrammatic) prototypes can be ranked from most abstract to most concrete or realistic:

1. conventional (fully abstract) content model
2. wire-frame mock-up
3. abstract layout diagram
4. low-fidelity paper prototype (rough-sketch)
5. high-fidelity paper prototype (realistic detail design)

Abstract Problems

Despite their demonstrated utility as a design tool, abstract prototypes have also proved to be stumbling blocks for some designers, especially relatively inexperienced ones or those who are still learning the rudiments of usage-centered design. The most common recurrent problems include:

- *difficulty naming or describing components in abstract terms*
- *difficulty distinguishing tools from materials*

- *difficulty translating abstract components into physical components*
- *difficulty laying out screens and other user interface contexts from abstract views*

Less-experienced designers often find it hard to think in abstract terms when naming the components to support task cases. Often they struggle to devise appropriately non-committal abstract names. Should it be called an "Employee Record" or an "Employee Record Holder" or an "Employee Description Holder" or what? Without careful choice of terms, designers may end up inadvertently incorporating implicit assumptions about what final form the components will take when realized in the actual user interface. An abstract component called "Employee Data Grid," for example, may imply a particular user interface data control.

For this reason, usage-centered design has discouraged the use of abstract component names that are too specific or incorporate technical terms or jargon. Instead of "Search Criteria Entry Field," for example, designers are encouraged to write something like "Sought-Person Description Holder." Unfortunately, although this practice defers the commitment to any particular implementation, if strictly followed, it introduces its own problems when the abstract prototype is later translated into an implementation model. If precise terms from the application domain, such as references to actual domain classes or methods, are abandoned in the content model in the interest of abstraction, the model—even though supposedly derived directly from the task model—can become disconnected from the other design models and the established vocabulary of the rest of the project.

Following recommended conventions for naming abstract components (such as, "Name Holder" "Constraint Stuff Getter" and the like), designers can ended up with a model that is not only disconnected from the final physical design, but also from the other models. These missing connections must ultimately be recovered and restored to the design in order to complete it and build it. (On one project, for example, the design team frequently had to go back and rediscover what domain objects were involved within each view before devising a visual prototype.) Content models with highly abstract components are also difficult to share with outsiders or with other parts of the development group: basically, if you were not there when they were developed, they may make little sense to you.

Beginning designers may also agonize over whether a particular need from the task model is best fulfilled by an active component or a passive one, that is, by an abstract tool or abstract material. Is an editable display field an active tool or a passive material? The debate goes on!

Bridging the Semantic Gap

Such difficulties aside, most designers, once they gain some practice, find that deriving an initial content model from task cases is usually relatively straightforward. Essentially, all that is necessary is to work through the task case narratives step-by-step, identifying the tools and materials needed on the user interface to enable the completion of each step. From a modeling perspective, the semantic gap between the task model and the content model is typically rather small, as can be appreciated from the example of Figure 4. There is usually a relatively simple, if not perfectly one-to-one, mapping from steps to abstract tools and materials.

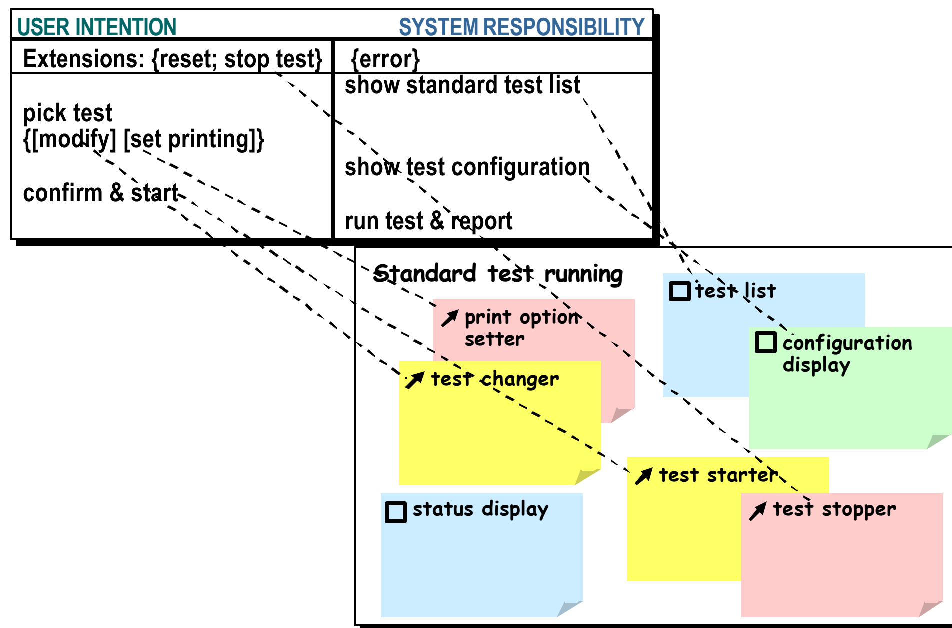

Figure 4 – Example of mapping from task case to content model.

In contrast, the semantic gap between a fully abstract content model and a good final design can often be enormous; the realistic paper prototype looks nothing like the content model. Between the abstract and the realistic models of the user interface lie dozens of decisions and difficult tradeoffs. Actual user interface components must be chosen or designed, screen layout must be determined, and other aspects of appearance and behavior must be resolved. Highly skilled, talented designers are usually able to leap this gap with relative ease, especially after gaining practice in usage-centered design, but beginners and more pedestrian designers often stumble and fall at this point. The result can be a significant amount of confusion and unproductive churning.

Although the abstract content model was originally devised to aid in the transition from a task model to a realistic prototype, experience suggests that it is relatively too close to the task model and too far from the desired goal of a finished visual and interaction design. Often, the abstract prototype as originally conceived has proved to be hard to develop relative to the payoff received.

Building a Better Bridge

A review of experiences on a range of projects has highlighted many of the disadvantages of abstract prototyping but has also convinced us of the overall advantage of a model intermediate between the task model and the final user interface design or implementation model. For relative novices, abstract prototypes appear to lead to substantially better initial designs than are typically achieved without their use. For advanced and sophisticated designers, abstract prototypes facilitate creative thinking, leading to more innovative solutions.

What is needed is a variant of the abstract prototype positioned closer to the final design so as to serve as a better translator from the task model to the implementation model. Such a revised form of model needs to be: (1) easier and more natural to develop in the first place, (2) easier to translate into an actual visual and interaction design; and (3) connected more effectively with the rest of the models through the

Domain Model. With respect to (3), an obvious solution is to ensure that descriptions and names used with abstract components employ the vocabulary of the domain and of the users, just as with all other models in usage-centered design.

To meet objectives (1) and (2), we concluded that abstract prototyping is best done in the form of an abstract layout diagram constructed from a standardized set of abstract components. For less experienced designers, a standard set of abstract tools and materials would simplify and guide the construction of the abstract prototype and would narrow the choices for a final design. For example, one could have a list of possible realizations of an abstract selector from which the designer could choose. For more advanced designers, a simple, standardized set of abstract components should speed and simplify the modeling process, freeing the designer to attend to subtle problems and creative solutions. Standardized abstract components should also make it easier to recognize and describe patterns that favor particular visual and interaction design solutions. Although we do not believe it desirable to turn the user design process into a cookbook approach, a standard way of describing abstract problems may help the design community to devise good general solutions to certain standard problems.

Canonical Abstract Components

Just as graphical user interfaces offer a standard toolkit of actual components from among which a designer can choose, canonical abstract components provide a standard "toolkit" of abstract components for abstract prototyping. The proposed set of canonical abstract components described here was devised through several iterations of successive refinement and trial application. The current version is by no means a theoretical minimum or rigorously defensible set, but we believe it is a practical and usable one that covers all the common cases arising in the practice of usage-centered user interface design.

Table 1 summarizes the set of canonical abstract components. (A printable full-page version of this table can be found at the end of this paper.) Canonical abstract components are identified by a name and a simple icon or glyph that serves as a graphic shorthand for advanced designers and as an aid to visual recognition and interpretation of abstract prototypes. (We are keenly aware that appropriate tool support is necessary to make such a shorthand a real shortcut.) The new symbols are derived by theme-and-variation from the two symbols already used to represent tools and materials, respectively. Although not all of the symbols may be intuitable on first sight, we have tried to make them serviceable as good reminders once they are learned.

The proposed canonical abstract components, summarized in Table 1, includes (a) generic or all-purpose abstract components (b) a core set of additional basic abstract components and (c) a handful of auxiliary, special-purpose components that were recognized as often desirable from a practical standpoint even if not theoretically required. (The optional components are marked with a double asterisk in Table 1.) All the materials are effectively specializations of the generic container and all the tools are specializations of the generic operation/action, so generic components can always be used for any purpose.

Table 1 – Summary of Canonical Abstract Tools and Materials

MATERIALS	DESCRIBED BY	EXAMPLES
container*	contents	Configuration holder
element (single item)	contents	Product image thumbnail
collection (multiple items)	contents [or set]	Personal address list
notification**	message/condition	Access privileges denied
accepter** (active material)	[Accept] contents	Search term entry field
TOOLS		
action/operator*	action	Print invoice
start	action	Start consistency analysis
stop/suspend	[action]	Stop searching
select	[Select] element	Group member selector
create	[Create] element	New customer
delete	[Delete] element	Remove network connection
modify	[Modify] element	Change shipping address
move	[Move] element	Put into approved list
duplicate	[Copy] element	Copy user profile details
go/link/drill**	[To/Open] target	To home page
perform** (and return)	[Perform] action	Set user preferences…
toggle**	[Toggle] condition	Detail display on/off

* generic (all-purpose) component ** optional (specialized) component

Materials

There are three basic abstract materials:

- container (generic)
- element
- collection

plus two auxiliary components:

- notification
- acceptor (active container).

For modeling, the *accept* tool can be thought of and used as either an active material, that is a container that takes input from the user, or as a tool operating on a container.

The proposed convention for naming abstract materials is simply to use the name of the contents, that is, the object, class, data element, or the like being represented. Appending a term like "Holder" or "Container" is acceptable if it clarifies the model, but is not required (e.g., "current machine configuration" or "target language holder"). For collections, the convention is either to use plurals to suggest multiple contents

(e.g., “special symbols”) or to describe the nature or type of collection (“address list”) as appropriate and required for clarity.

Whether on paper, in CASE tools, or on pre-printed forms, abstract components can be labeled with either the icon alone or the icon plus component type followed by the user-supplied name. For example:

Notification is actually just a message container or indicator and should be named by the message or condition or event represented (for example, “too many items” or “machine not synchronized” or “on”).

Tools

Operations and actions are two distinct kinds of abstract tools. Operations are abstract tools that operate upon materials, and actions are abstract tools that cause or trigger some action. In addition to a generic action/operation, there are eight basic abstract tools.

actions:

- *initiate/start*
- terminate/quit

operations

- select
- create
- delete
- modify
- move
- duplicate

auxiliary tools:

- go/link/drill
- perform (with return)
- toggle

The proposed convention for naming abstract tools is just to specify the action. For generic actions, the prefixes “Do” or “Start” are optional (e.g., “Do symbol checking” or “Print symbol table”). The graphical symbol or name alone can be used where the results are clear for the intended purposes of the model. The assumption is that the graphical symbol and action name could, in most cases, substitute for each other. Thus, the following are the same component described in three different ways:

Close configuration

Where needed for clarity, tools that operate upon materials (operations) should name the materials. In abstract prototypes, operations can just be placed with or on the

materials upon which they operate wherever this is graphically convenient and meaningful.

What does this all look in practice? We now believe that for most user interface design, the most useful form of abstract prototype is an abstract layout diagram in which the size and relative positions of abstract components is meaningful. Figure 5 is an example of an abstract layout using canonical components. It is intended to be illustrative rather than exemplary or worthy of imitation. As shown in this example, nesting of abstract components within other components is sometimes necessary or expedient.

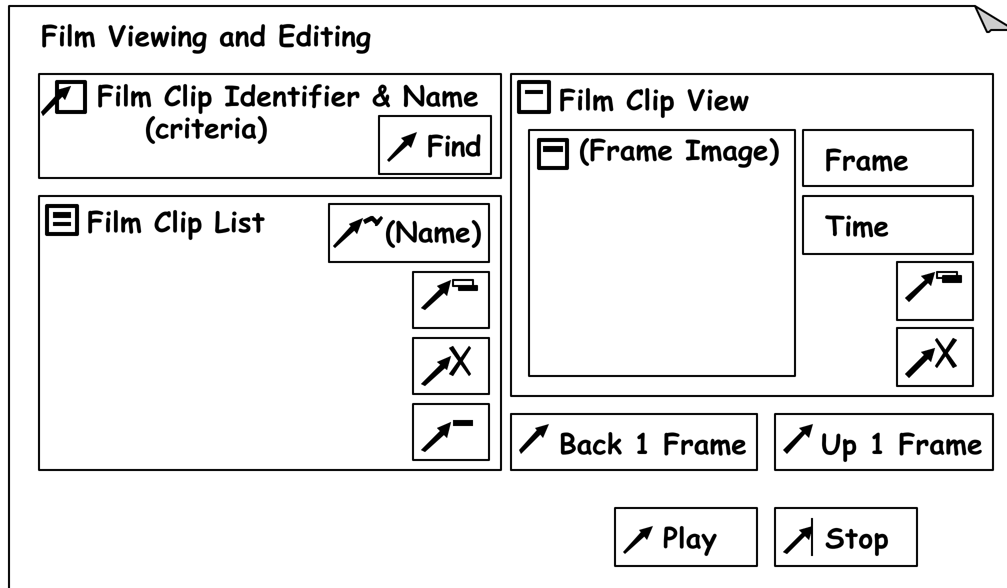

Figure 5 – Example of abstract layout using canonical components.

As already discussed, varied forms of abstract prototypes have varying degrees of abstraction. Another variation that is often useful, especially for Web-based projects or for extremely large design problems, is a text-based, non-graphical form that merely lists the views and their contents. As can be seen in the example below (a simple translation based on Figure 5), the use of canonical abstract components, with or without graphic symbols, improves readability and aids interpretation in text-based content models. (Note the nesting of abstract components.)

Context: **Film Viewing and Editing**
- *Accept:* **Film Clip Identifier and Name**
  - *Do:* **Find**
- *Collection:* **Film Clip List**
  - *Modify:* **Name**
  - *Duplicate*
  - *Delete*
  - *Select*
- *Element:* **Film Clip View**
  - *Element:* **Frame Image**
  - *Element:* **Frame**
  - *Element:* **Time**

Duplicate
Delete
Do: **Back 1 Frame**
Do: **Up 1 Frame**
Do: **Play**
Do: **Stop**

Designing from Canonical Prototypes

Design Process

The use of a set of canonical components in abstract prototypes offers potential advantages for both highly experienced, sophisticated designers and designers of more modest experience and talents. The translation from canonical prototype to realistic prototype or final design involves two concurrent and interdependent design activities: visual design and interaction design. Visual design involves (a) the selection or design of visual components to realize each canonical component combined with (b) the layout of these visual components within a view or context in the interface. Interaction design involves (a) the selection or devising of interaction idioms, (b) describing the required behavior of the interface and underlying system, and (c) organizing the workflow or sequence of interaction within and between views or contexts.

For conventional designs, each canonical component is simply realized by one or more standard user interface widgets. For the best results, the designer should identify the various alternative realizations for each abstract component. A trial layout for the paper prototype is constructed based on an initial selection from among these alternatives. The selection of actual user interface components typically implies much of the interaction design and the layout determines the workflow. The resulting interaction design should be reviewed against the task cases being supported and refined, along with the component selection and layout, for efficient and effective support of task cases.

Where high performance or breakthrough design is the objective, the canonical model provides additional guidance for creative design. The use of canonical components makes it easier for experienced designers to recognize and categorize patterns or common situations that imply certain kinds of problems or solutions. For example, a review of past design work suggests that certain configurations are often ripe with the opportunity for invention of new user interface controls that are both highly efficient to use and make better use of screen real estate. In the hands of an advanced design team, for example, nested combinations of containers, collections, or elements with included (nested) tools can often be turned into compact and efficient non-standard user interface controls.

In outline, the process of innovative design based on a canonical prototype goes like this. First, closely related or grouped abstract components, especially nested combinations, are noted. For each such group or combination:

6. Identify both conventional/routine realizations and creative ones.
7. Select promising combinations.
8. Synthesize and refine.

Applied Example

For a simplified example, consider the portion of an abstract prototype shown in Figure 6, in which a collection of items can be arbitrarily rearranged into a new order by the user. A number of visual design approaches are possible, including, among others:

- *a list with editable sequence numbers*
- *a list with up and down buttons that move a selected item within the list*
- *a temporary holding list allowing items to be removed and reinserted within the main list*

Each of these approaches involves slightly different visual components and layout issues.

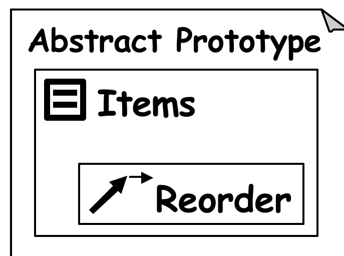

Figure 6 – Example with tool nested in container.

Interaction design involves identifying potential interface behavior and interaction idioms to solve this problem. In this case, these include, among others:

- *click to select source point and target*
- *drag-and-drop*
- *edit sequence numbers*
- *click on move-up or move-down buttons*

A promising combination that supports both novice and more advanced usage patterns would be to support both moving within the list using up-down buttons and moving by drag-and-drop. An initial design might resemble Figure 7(a), which highlights several problems. The up-down buttons are easily confused with the scrollbar buttons if placed in their conventional location. If simply moved to the left, however, they are easily missed and their function might not be clear.

Such problems can be overcome with appropriate visual and interaction design. As shown in Figure 7(b), the distinctiveness of the up-down buttons can be enhanced by shifting them to the left, changing their shape, and highlighting them with color glyphs. The interface can become self-instructive through progressive enablement if the up-down buttons are initially disabled (grayed) and become enabled and highlighted with color on selection of an item in the list. Being able to drag-and-drop items within the list is a desirable capability, especially for advanced users, but the capability is hidden behavior without suitable feedback to the user. Move affordance can be communicated to the user by changing the cursor to an up-down move form on mouse-down whenever an item in the list is being selected.

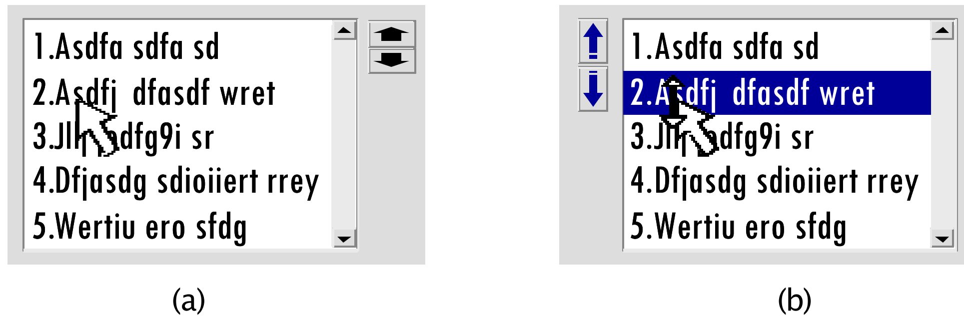

Figure 7 – Refinement of possible solution to problem.

Conclusions

Abstract layout diagram using canonical abstract components offer an exciting new tool to smooth and speed the process of usage-centered design. As described here, these “canonical prototypes”

- *are constructed from specific abstract components selected from a small set of standard components described in a standard notation*
- *show layout, including relatives size, position, and nesting or overlay of components*
- *use the same standard vocabulary of the users and application domain as employed all other models throughout a project*

Canonical prototypes are, thus, simple and straightforward to construct from task cases and yet are also closer to a final design than previous forms of abstract prototypes. Canonical prototypes allow designers to easily model the specific contents of user interfaces and to experiment with general layout without committing to details of appearance or graphic design. The designer is provided with a complete set of standard but abstract components from which to choose in expressing the content and general layout of user interface designs. Because the resulting models more closely resemble actual user interfaces while omitting detail, canonical prototypes facilitate final design without closing out the possibility of inventive or non-standard realizations.

Future work could further enhance the value of canonical prototypes. For any particular implementation environment, the various available realizations can be cataloged in advance for each different canonical component. Especially for beginners, such guidance could be very useful. For more advanced designers, user interface design patterns can be organized and described in terms of combinations or configurations of canonical components.

References

Constantine, L. L. (1998) “Rapid Abstract Prototyping,” *Software Development, 6,* (11), November.

Constantine, L. L. (2000) “Inventing Software,” *Software Development, 8,* (5), May.

Constantine, L. L., & Lockwood, L. A. D. (1999) ***Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.*** Boston: Addison-Wesley.

Glossary

abstract layout	a low-fidelity prototype of a user interface view that shows the layout, including the relative size and position of components, but not the exact details of appearance
abstract material	an abstract user interface component representing a container, information, or data
abstract prototype	any of a range of models representing a user interface design in the abstract
abstract tool	an abstract user interface component that operates upon material(s) or initiates some action(s)
canonical component	one of a standard set of abstract tools and materials
canonical prototype	an abstract prototype showing layout, size, and position of canonical components described in terms of the vocabulary of users and of the application domain
content model	an abstract prototype representing only the abstract tools and materials in a view independent of their appearance, behavior, or layout
task case	an essential use case [Constantine & Lockwood, 1999] supporting one or more user roles
view	an interaction context; a portion of a user interface within which a user can interact with a system; for example, a screen, dialog box, window, or the like

Notes

[1] Some terminology in usage-centered design is in the process of being revised for simplicity and economy of expression as well as for compatibility with other methods and notations. See Glossary.

Learn more about usage-centered design, including training in canonical abstract prototypes, at http://www.forUse.com.

Constantine & Lockwood, Ltd.

Canonical Components for Abstract Prototypes

© 2001, Constantine & Lockwood

MATERIALS	DESCRIBED BY
container*	contents
element (single item)	contents
collection (multiple items)	contents [or set]
notification**	message/condition
accepter** (active material)	[Accept] contents

TOOLS	
action/operator*	action
start	action
stop/suspend	[action]
select	[Select] element
create	[Create] element
delete	[Delete] element
modify	[Modify] element
move	[Move] element
duplicate	[Copy] element
go/link/drill**	[To/Open] target
perform** (and return)	[Perform] action
toggle**	[Toggle] condition

EXAMPLES

- Configuration holder
- Product image thumbnail
- Personal address list
- Access privileges denied
- Search term entry field
- Print invoice
- Start consistency analysis
- Stop searching
- Group member selector
- New customer
- Remove network connection
- Change shipping address
- Put into approved list
- Copy user profile details
- To home page
- Set user preferences…
- Detail display on/off

* generic (all-purpose) component ** optional (specialized) component