



Canonical Abstract Prototypes for Abstract Visual and Interaction Design

Larry L. Constantine
University of Technology, Sydney, (Australia)
Constantine & Lockwood Ltd | lconstantine@foruse.com

Abstract. Abstract user interface prototypes offer designers a form of representation for specification and exploration of visual and interaction design ideas that is intermediate between abstract task models and realistic or representational prototypes. Canonical Abstract Prototypes are an extension to usage-centered design that provides a formal vocabulary for expressing visual and interaction designs without concern for details of appearance and behavior. A standardized abstract design vocabulary facilitates comparison of designs, eases recognition and simplifies description of common design patterns, and lays the foundations for better software tools. This paper covers recent refinements in the modeling notation and the set of Canonical Abstract Components. New applications of abstract prototypes to design patterns are discussed, and variations in software tools support are outlined.

Usage-Centered Design

Usage-centered design is a model-driven approach to the presentation design and interaction design of software [1] and Web-based applications [2]. It is a robust and adaptable process with a proven record of success over nearly a decade of application on a wide variety of projects ranging from small, XP-style applications programming [3] to large-scale industrial tools development [4]. It has proved particularly effective for complex problems in which the efficiency and dependability of user performance is critical, such as in medical informatics [5] or industrial automation applications programming, where it has led to radical improvements in user task and problem-solving performance with award-winning results [4, 6].

In usage-centered design, user interface designs derive directly and systematically from a series of core models. The final presentation and interaction designs are based directly on models of interface contents, which derive in a straightforward fashion from models of user tasks, which are based in turn on models of the roles users play in relation to the planned system. The core of the process is a robust, fine-grained task model comprising a collection of interrelated use cases expressed in so-called essential form [7, 8]. The close fit of the user interface design to this task model can yield dramatic improvements in ease of learning as well as more reliable and efficient user task performance.

Draft preprint. To appear in Joaquim Jorge, Nuno Nunes, and João Falcão e Cunha (eds.) *Proceedings of DSV - IS'2003 - 10th International Workshop on Design, Specification and Verification of Inter-active Systems, LNCS - Lecture Notes in Computer Science*. Berlin: Springer-Verlag.

Usage-centered design was originally developed and has continued to evolve driven primarily by pragmatic concerns with supporting the design process of practicing designers working on real-world projects. The objective has always been to facilitate a design process that is simultaneously efficient, reproducible, and creative. The models it incorporates were devised to provide the most conceptual and creative leverage for the least amount of effort on the part of analysts and designers. To this end, usage-centered design differs in both philosophy and practice from many mainstream user-centered design approaches [2, 9]. For example, although task models in one form or another are elements of many design processes and are widely used in practice, the task cases employed in usage-centered design are specifically constructed to be the simplest and most compact expression of a given set of tasks, thereby promoting the identification of simpler designs to support those tasks. Similarly, and unlike more elaborate methods such as the Rational Unified Process [10, 11], usage-centered design seeks to reduce the number and complexity of design artifacts to the absolute minimum required for an orderly and effective process.

Content Models and Modeling

Content models represent the contents of user interfaces and their various constituent sections or parts independent of details of appearance and behavior. Content models thus abstract from more realistic representations of user interface designs, such as the paper prototypes or mockups commonly sketched by designers, and for this reason, they are sometimes referred to as abstract prototypes [12]. As abstractions, they can serve as an intermediate bridge between task models and realistic designs, smoothing, simplifying, and systematizing the design process. Content models help clarify and define what a user interface must contain and how its contents are partitioned before its design is worked out in detail. They encourage reasoning and experimentation with how component parts of a user interface are combined and distributed to form a coherent, understandable, and usable whole.

User interface prototypes can be arrayed on a continuum of abstraction. The simplest and most abstract content models, called content inventories, consist of simple lists inventorying the information and controls to be collected within a given interaction context, such as, a window, dialog box, page, or screen. The visual content inventories using sticky notes that were first introduced in usage-centered design [1, 12] also incorporate position or spatial relationship among interface contents. So-called wire-frame schematics outline (with "wire frames") the areas occupied by the various interface contents. Figure 1 illustrates a single Web page as described by a content inventory and a wire-frame schematic. At the most realistic and least abstract end of the spectrum are low-fidelity paper prototypes or rough sketches, high-fidelity paper prototypes, and, finally, accurate mockups or even working or partially functional simulations.

The more abstract models facilitate solving problems in user interface organization, navigation, or overall architecture, leaving aside the details, while realistic prototypes help resolve detail design decisions in layout, visual presentation, and component selection, as well as fine points in interaction design and interface behavior. Indeed, skilled, disciplined designers tend to work from higher-level abstract representations toward progressively more realistic and detailed representations as the design evolves and is refined.

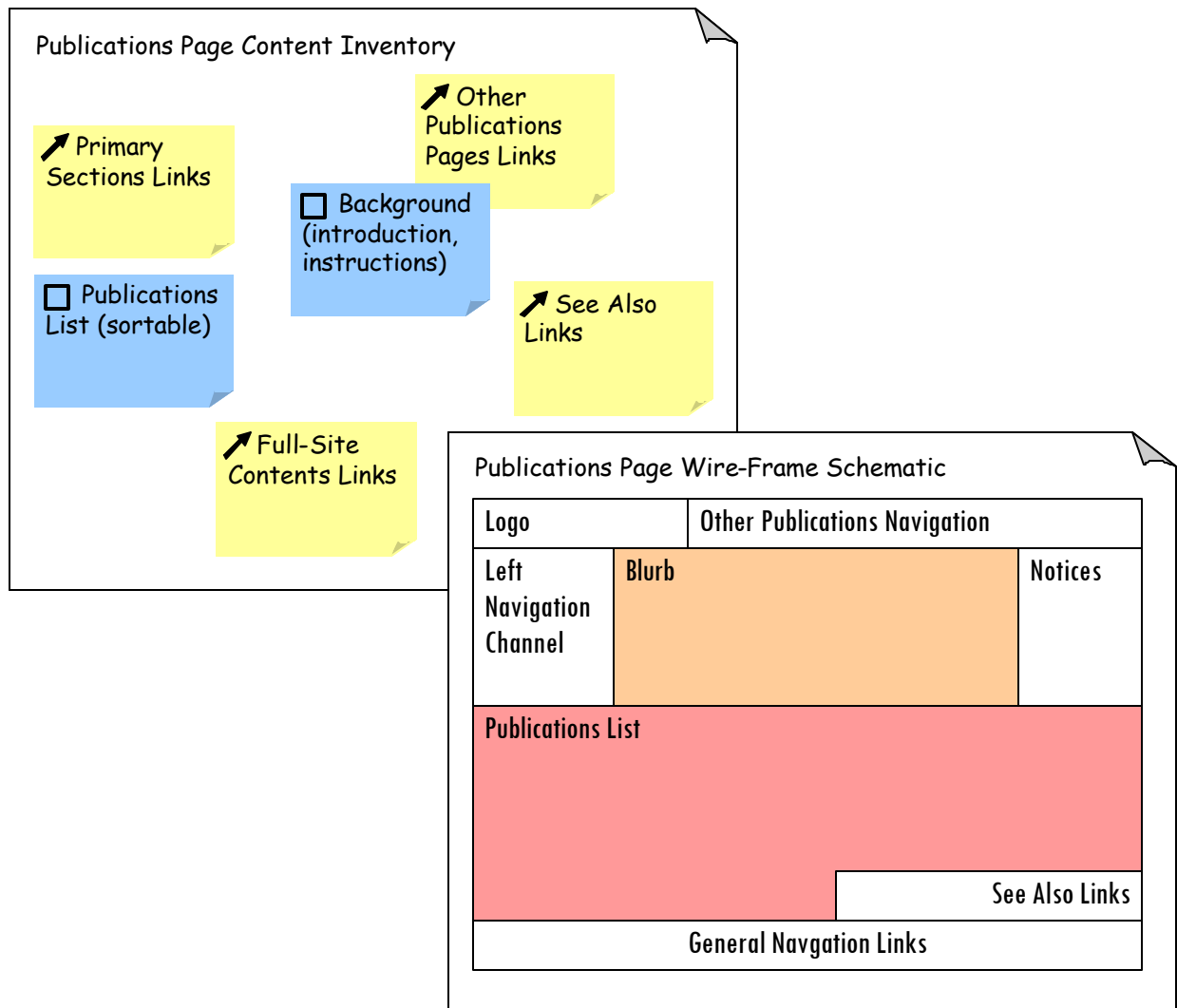


Figure 1 - Sample visual content inventory and wire-frame schematic for Web page.

Canonical Abstract Prototypes

Canonical Abstract Prototypes are a model specifically created to support a smooth progression from abstraction toward realization in user interface design. The impetus for developing them arose from a growing awareness among practitioners of usage-centered design regarding the substantial conceptual gap between the task models needed to drive an effective design and the detailed, realistic prototypes needed for successful implementation. Particularly on large projects, the need for some intermediate form of representation became acutely apparent. Simple content inventories had proved both too abstract and too imprecise for resolving design issues in very complex user interfaces.

Canonical Abstract Prototypes emerged from a workshop of practitioners convened in 2000 by Constantine & Lockwood, Ltd. [13]. Superficially, Canonical Abstract Prototypes, such as the one illustrated in Figure 2, resemble wire frame schematics but are constructed from a standardized set of universal abstract components. Each Canonical Abstract Component has a specific abstract interactive function, such as toggling a state, creating information, or providing a notification. These standard

interactive functions are represented by simple symbols. Canonical Abstract Components model not only the interactive functions to be provided by a user interface, but also the position, size, layout, and composition of the user interface features. The notation, indeed the entire scheme for Canonical Abstract Prototypes, was devised so as to promote precise and consistent modeling while maximizing the utility for practicing visual and interaction designers working on real-world projects.

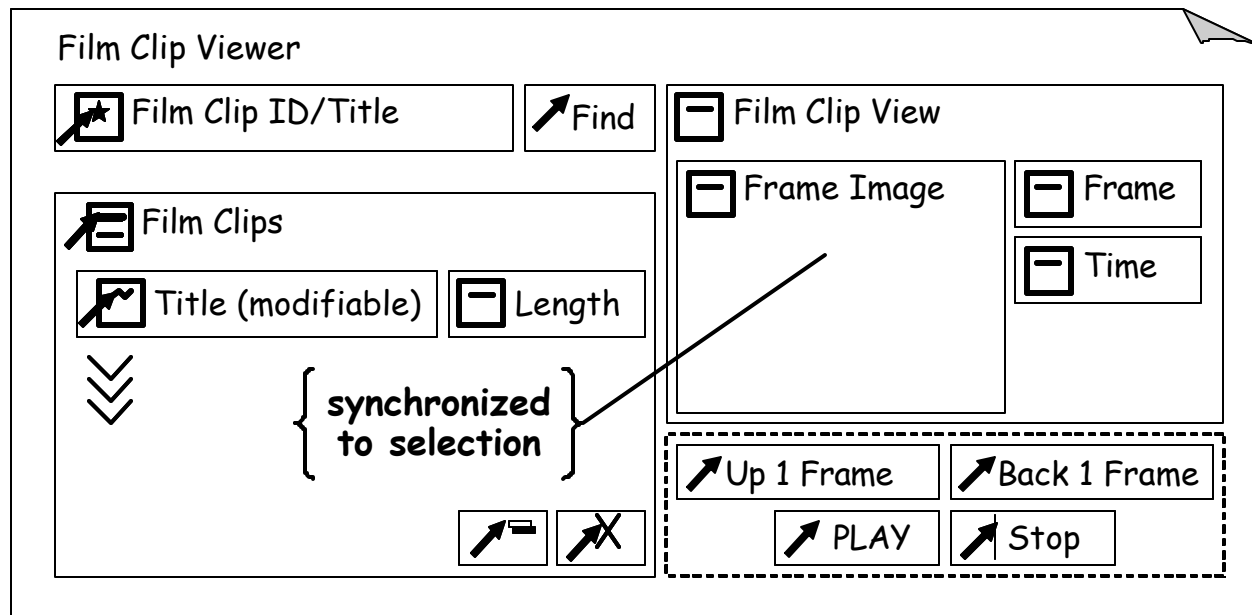


Figure 2 - Example of a Canonical Abstract Prototype show examples of key notational elements.

The use of a standard set of abstract components serves a number of purposes. Easy selection from a palette of available components can speed and simplify the process of abstract prototyping. The standard form facilitates comparisons of designs and makes it easier both to recognize and to describe recurring patterns or common configurations. In addition, because the abstract components are related by their specific interactive function to particular realizations, Canonical Abstract Prototypes provide direct guidance for the visual and interaction design of the user interface.

The notation was worked out with careful attention to usability, an often under-appreciated aspect of modeling schemes [14]. The objective throughout has been to make it easy for even inexperienced designers to interpret the diagrams and infer the meaning of the notation. Although other approaches have attempted to achieve precision and consistency in modeling user interface contents with standard notations, notably UML (see, for example, [15, 16, 17, 18]), the resulting accommodations and compromises, such as using class models for representing interface contents and content navigation, are often visually awkward and may be particularly unhelpful for visual designers.

Other models and notations, such as, Abstract User Interfaces [19], have been devised for specifying user interfaces in order to support of "plasticity" or the ability to deploy user interface implementations in multiple target environments. In contrast, Canonical Abstract Prototypes were devised for designing user interfaces in order to support design decision making at a higher level of abstraction than typical paper prototypes.

Like other related techniques, Canonical Abstract Prototypes leave open many details of implementation or realization, but the intention is that these details will ultimately be resolved by a designer concerned with human use and performance rather than by a rendering engine or deployment process.

Canonical Abstract Components

Canonical Abstract Components model the various interactive functions—such as interrupting an action or displaying a collection of information—needed within the realized user interface. Each canonical abstract component has a specific interactive function represented by a symbolic graphical identifier and a descriptive name. The graphical symbols serve as an easily learned shorthand for the various interactive functions. The notation is built on two universal symbols—a generic tool or action and a generic material or container—plus extensions and combinations of symbols.

The effective support of work of any kind requires the assembling and organizing of collections of tools and materials needed for the performance of particular tasks or closely related sets of tasks [20]. In software or Web-based applications, materials are the containers, content, information, data, or other user interface objects operated upon or manipulated in the course of task performance. Examples include the body of an email message, a count of occurrences, or a drawing shape. Tools are the actions, operators, mechanisms, or controls that can be used to create, manipulate, transform, or operate upon materials. Examples might include a color selector, a copy command, or a button to increment a count. In practice, many user interface features exhibit characteristics of both and may be thought of as hybrids that are simultaneously containers and active controls. A ubiquitous example is a text entry box that serves both to hold or display information and to enter or manipulate it.

The symbolic notation reflects these fundamental distinctions and is built from two universal symbols or glyphs: a square to represent a container and an arrow to represent a tool—with the combination of the two representing a generic hybrid component, as shown in Figure 3. The generic container can be used as the abstract representation of any kind of material whatsoever, the generic action can represent any sort of tool, and the generic hybrid can represent any component with characteristics of both. Specializations having more specific interactive functions are constructed as logically consistent and transparent combinations and elaborations of the generic components. Thus, for example, a collection is represented by the symbol shown in Figure 4a, a selection tool by the symbol in Figure 4b, and a selectable collection by the symbol in 4c.

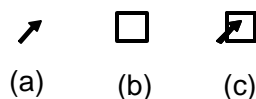


Figure 3 - Basic symbols for Canonical Abstract Components showing: (a) generic abstract tool, (b) generic abstract material, (c) generic abstract active material.

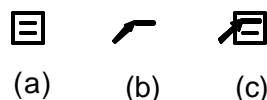














Figure 4 - Extension and elaboration of basic symbols for Canonical Abstract Components illustrated with: (a) abstract collection container, (b) abstract selection tool, (c) abstract selectable collection material.

The original set of abstract components and their symbolic representation have been refined and extended through feedback from many professional practitioners. (Over 25,000 copies of the working report [13] and draft template have been downloaded.) The current version of the notation contained in this paper incorporates a stable collection of 21 components: 3 generic components, 11 specialized abstract tools, 3 specialized abstract containers, and 7 abstract hybrids. In addition to the abstract components themselves, the notation includes mechanisms for annotation (curly brackets), conceptual grouping (dashed outlines), and repetition of interface elements (triple chevron), which can be seen illustrated in Figure 2. As with the original concept, the objective continues to be to strike a workable balance between conceptual simplicity and providing a versatile toolkit with a wide variety of specific interactive functions of demonstrated utility and importance.

Table 1 - Canonical Abstract Components: abstract tools

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary

Canonical Abstract Components are modeled using a rigorously consistent, extensible notation. Tables 1 through 3 detail the complete set of Canonical Abstract Components along with examples. As the dozen abstract tools in Table 1 illustrate, interactive functions are distinguished from the perspective of users in interaction with a user interface. In this they are distinguished from Abstract Interaction Objects (AIO), another intermediate abstraction sometimes employed in user interface design and development [21]. Whereas an AOI corresponds to or is an abstraction from a specific kind of user interface component, such as a drop-down selection list, each CAC corresponds to a particular interactive function from the perspective of users.

Table 2 - Canonical Abstract Components: abstract materials












SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
	notification	Email delivery failure, Controller status

Table 3 - Canonical Abstract Components: abstract hybrids or active materials

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	active material*	Expandable thumbnail, Resizable chart
	input/accepter	Accept search terms, User name entry
	editable element	Patient name, Next appointment date
	editable collection	Patient details, Text object properties
	selectable collection	Performance choices, Font selection
	selectable action set	Go to page, Zoom scale selection
	selectable view set	Choose patient document, Set display mode

From Task Model to Design

In practice, most user interface design processes depend on a certain amount of hand waving and unexplained creative leaps to cross the gap from models or concepts to realizable designs. Even when task models of one sort or another are employed, the activity of creating an initial paper prototype sketch might often appropriately be labeled "Magic Happens Here."

Using Canonical Abstract Prototypes, the transition from task model to design is reduced to two relatively straightforward translation processes, each of which addresses a limited set of specific decisions and issues. Although the design process is thus made substantially more orderly and manageable, the role of creative problem solving has not been eliminated—systematization is not equivalent to mechanization. In contrast with schemes for automatically generate user interfaces from specifying models [22, 23], canonical abstract prototyping recognizes the pivotal role of human creativity and invention in designing good user interfaces. The use of canonical abstract prototypes merely serves to focus the attention and creative energies of the designer on matters of importance in the design task at hand.

The process by which the initial abstract prototype is generated from the task model is roughly as follows. For each cluster of closely related task cases, an interaction context is provisionally defined. For each task case in a cluster, the defining dialog or narrative is examined [8]. For each step in the narrative, the interface contents necessary for performance of the step are identified and appropriate abstract tools and materials are added to the contents of the interaction context. In practice, this is often accomplished using sticky notes to represent abstract components (as shown in Figure 1). Where

specialized Canonical Abstract Components are clearly needed or preferred, these are used; otherwise, generic tools, materials, or combinations are used.

Once all the necessary tools and materials have been incorporated into the interaction context, the layout and organization of the interaction context are explored. Area is allocated based on such things as importance, complexity, and user focus. Canonical Abstract Components are positioned according to such issues as workflow and conceptual or semantic interrelationships and may be by combined into composite components when meaningful and potentially useful.

The ultimate result of the abstract prototyping process is a complete but abstract provisional design for the entire user interface. Such an abstract prototype can be validated against the task model by walking through task cases or scenarios to verify that tasks can be performed with reasonable efficiency given the tools and materials available. Inefficiencies arising from navigation among interaction contexts or the absence of certain optional tools or materials are readily discovered at this stage.

Generating a realistic prototype based on an abstract prototype is, at least in principle, another relatively straightforward activity. For each Canonical Abstract Component or combination of closely connected components, an effective realization is selected from among the actual alternatives available in the target user interface environment. One of the advantages of abstract prototyping is that it narrows the choices of detail design, since any given canonical abstract component has only a certain number of plausible realizations within any given implementation environment and visual vocabulary. Thus, for example, an abstract Toggle could be realized in HTML using a checkbox, a pair of radio buttons, or a two-item selection list.

The choice of realization depends on tradeoffs in available screen real estate, effectiveness of presentation, efficiency of interaction, and the like. In many cases, the need for or advantages of custom, original user interface components with novel appearance and/or behavior becomes apparent from the abstract prototype, which can then serve as a guide for designing the new components should this be the preferred option. In the process of selecting or designing particular realizations, layout and organization can, of course, be affected and may need revision. For example, Figure 5 illustrates a Canonical Abstract Prototype for a portion of an intranet site supporting the browsing of past issues of a corporate newsletter. Figure 6 shows a mockup of a creative realization that goes beyond a simple one-to-one translation by integrating closely related controls and providing synchronized views for easy and versatile browsing.

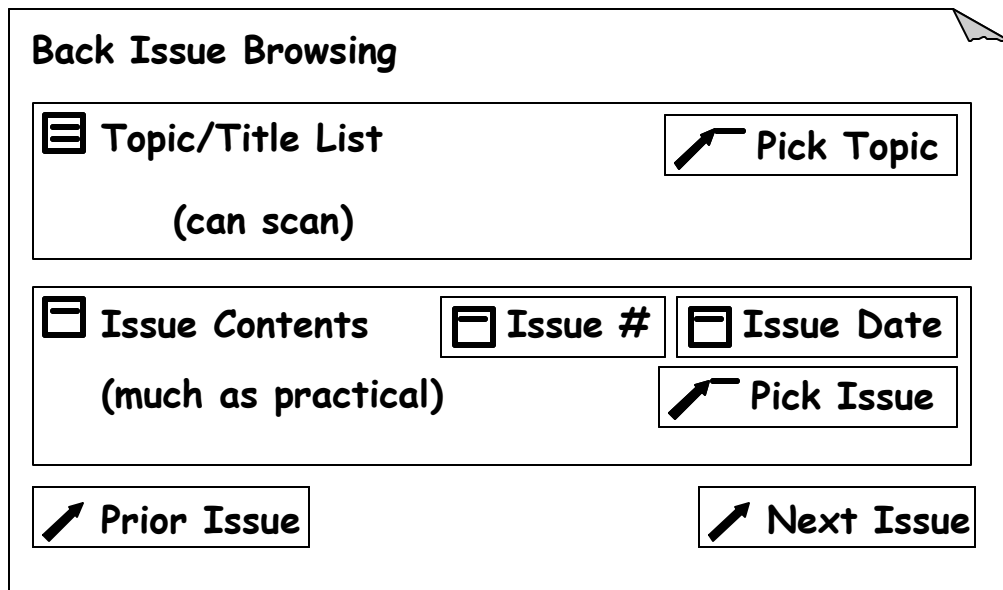


Figure 5 - Example of a Canonical Abstract Prototype for browsing back issues of a corporate newsletter by topic, issue number, or date.

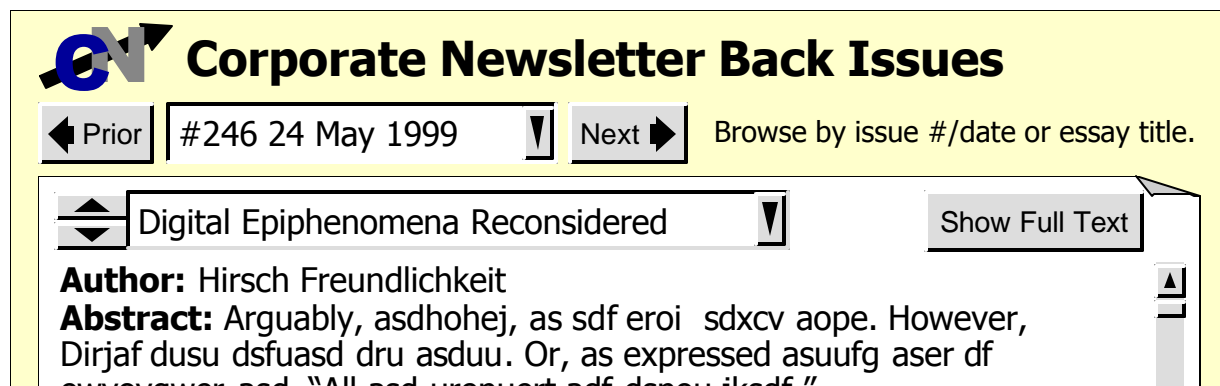


Figure 6 - Example of design mockup for Canonical Abstract Prototype shown in Figure 5 with custom controls for stepping through and selecting issues and topics with synchronized contents, issues, and topics

Abstract Design Patterns

The ability to express design patterns in terms of generalized abstract models has been a staple of the patterns movement in software engineering from the outset but has seen little use in interaction and user interface patterns, in part because of the absence of appropriate modeling schemes. The original working document on Canonical Abstract Prototypes [13] highlighted their potential as a medium for both the identification and the expression of high-level design patterns for user interfaces. Recent work in this area includes using Canonical Abstract Prototypes to define and describe selected non-obvious design patterns of broad utility. The objective is to capture and model subtle issues in visual and interaction design in which it is possible to identify best-practices or preferred resolution of competing design forces in an abstract, generalized form.

Detail View Navigation Pattern

The role of the notation in elucidating abstract design patterns can be illustrated with the Detail View Navigation pattern [24]. This pattern is related to some other published patterns, notably Guided Tour and Hybrid Collection [25], but differs in modeling more fully generalized "best-practices" solutions. Detail View Navigation is briefly described here in the more or less conventional style in terms of the problem, issues (or forces, as they are sometimes called), and the solution.

Problem. How can the user interface support flexible, efficient, and convenient exploration by users of a collection of items available both as a list or other ordered collection and as individual items in detail or expanded views?

Issues. The user wants as much information as possible about each item to understand it or to make a decision, but limited screen real estate and/or complex items make it problematic to present all information on all items together in one visual context. A list view makes it easy for the user to scan successive items but presents limited information. A detail or expanded view for each item provides more complete information but does not in itself facilitate movement or scanning among items.

In order to see the next item in a detail/expanded view, the user typically must return to the list view, visually relocate the just-viewed item in the list, then drill-down on the following item to bring up its detail/expanded view. Particularly when the details of many items are potentially of interest, the process is awkward, inefficient, and prone to error because the user can too easily click on the wrong item in the list view.

Solution. The resolution of these issues is to provide controls on each detail/expanded view for navigation directly to the preceding and succeeding (and possibly other) detail/expanded views. The Canonical Abstract Prototypes shown in Figure 7 represent the visual organization of the list view and detail/expanded view conforming to this pattern. Optionally, as space permits a condensed description or short abstract can be included in the list view to further facilitate user scanning and selection.

In the detail/expanded view, direct access to the list view is arranged with the forward and back functions because these are all logically related and belong at the likely focus of attention when the user is considering where to look next. If it is preferable to have the forward and back navigation functions adjacent to each other for any reason, the most logical position for access to the list view would be just above them. The redundant navigation controls included both above and below the contents of the detail/expanded view conform to another abstract design pattern: Top-and-Bottom Navigation.

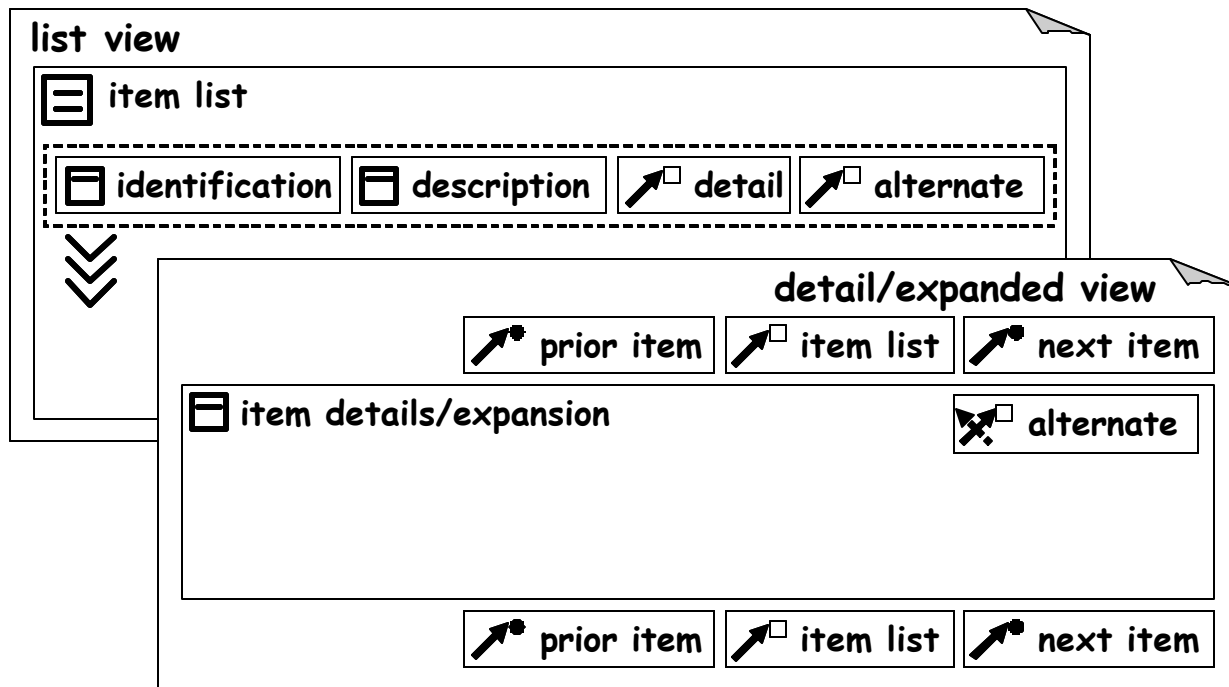


Figure 7 - Canonical Abstract Prototype illustrating the resolution for the Detail View Navigation pattern.

In some cases, an item might be available in more than one detail/expanded or in alternate formats. For example, a paper could be available in both HTML or Adobe Acrobat formats, or as either an abstract or full text. In such cases, access to the alternate views or formats should be surfaced—made directly available—in the list view as well as in the detail/expanded view, where it ideally functions as a toggle if there are only two views.

An item identifier element associated with the prior and next navigation controls is optional, as shown in the abstract prototype in Figure 8. It serves as a reminder and to enrich the context for the user, making it more likely that the user will navigate effectively. The identifier could be the name, number, date, or compressed title of the adjacent item. Screen real estate, implementation problems, and increased complexity to the user must be weighed against the potential value of the added information. Other variations and extensions are covered in the full pattern [24].

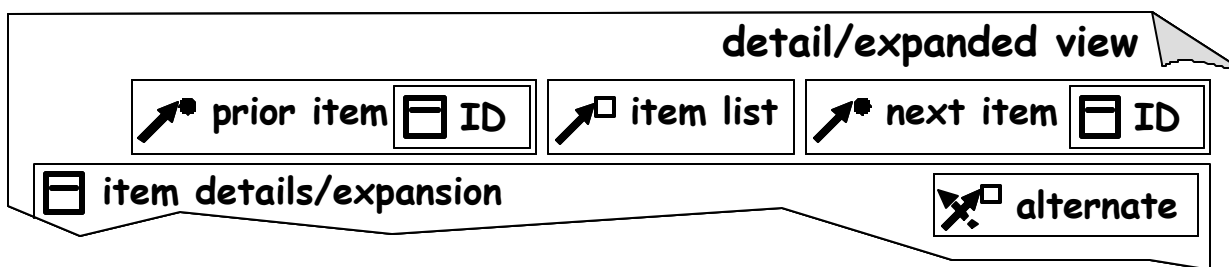


Figure 8 - Canonical Abstract Prototype illustrating an elaborated resolution for the Detail View Navigation pattern.

Application. This pattern applies to any ordered collection of items that can be viewed in either list or collected form or as individual items in a detail or expanded view.

Representative examples include back issues of a periodical or serial publication (such as illustrated in Figure 9), papers in a series, as well as chapters in a book or articles within an issue of a journal or other publication. For a single issue of a journal or book, the list view is the table of contents and the detail views are the individual articles or chapters. Neither the collection nor the items need necessarily be in text form. For example, the pattern applies to a collection of thumbnail images that can also be viewed in enlarged format.

If each article or chapter is structured as multiple pages, controls for direct article-to-article or chapter-to-chapter navigation need to be supplied independent of forward and back paging. Paging forward beyond the end of an article or chapter should take the user to the first page of the next article or chapter, and paging backward from the beginning should take the user to the last page of the previous article or chapter, exactly as in a magazine or book and in keeping with user expectations.

This pattern can also apply to search results, such as in e-business on the Web, enabling the customer to step through successive product descriptions without having to repeatedly bounce back to the search results page. Similarly, a customer should be able to step through descriptions of successive products within a product category without having to jump back up a level in the product hierarchy. (See [24] for further discussion and examples.)

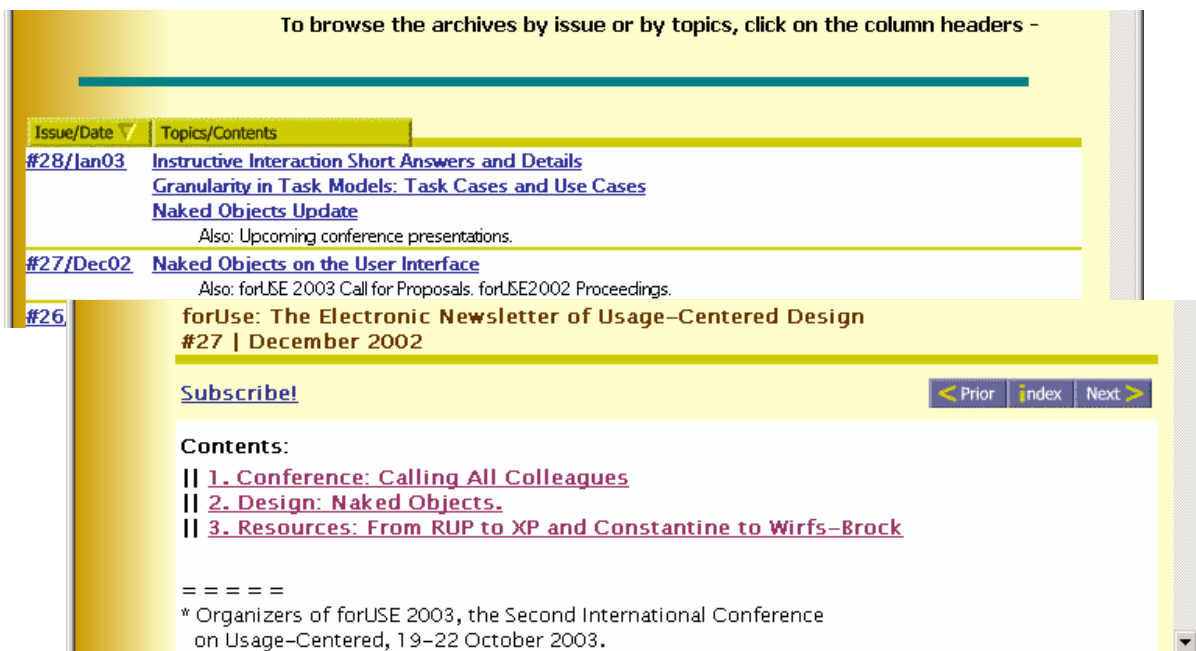


Figure 9 - Example of the Detail View Navigation pattern as realized in a newsletter archives on a Web site (www.foruse.com).

Tool Support

Although the notation has primarily been used for abstract prototyping with sticky-notes, whiteboards, and other "low-tech" tools, software tool support is ultimately needed for wider application and realization of the full potential of Canonical Abstract Prototypes. Several levels of tool support can be identified. At the lowest level, is a specialized drawing tool incorporating a palette of canonical abstract components that can be labeled, positioned, and resized within a drawing space representing an

interaction context. Supplemented by an overview representing how multiple interaction contexts are interconnected, such a simple stand-alone tool, even in the absence of model semantics or rich behavior, would have significant value in facilitating high-level and architectural design of complex software and Web applications.

At a more advanced level of sophistication, a specialized drawing tool would be integrated into support for the usage-centered design process by linking the content model back to the task model. Such a tool would enable the designer to move between interaction contexts and the task cases they support or between steps in a narrative and the abstract components supporting the steps.

At a yet higher level, the abstract prototype would be linked to a realistic representational prototype expressed in a conventional drawing tool. For real value, the linkage must be more than a connection from an abstract interaction context to a screen design; features as drawn in the "paper prototype" need to be linked with particular abstract components in the abstract prototype.

At the highest level, the content model would also be linked forward all the way to the actual user interface, so that abstract components could be mapped to real software GUI components. Ideally, the connection would be created transparently as part of using a conventional visual development environment. The associations would need to be maintained so that designers and software engineers could trace forward from task cases to abstract contents to actual code and from GUI features as designed and programmed back to steps in a task narrative—with or without a stopover in a realistic paper prototype.

Some might argue that the ultimate in software tool support is to enable automatic generation of the actual user interface and its code directly from abstract models of the problem and this is the explicit objective in some model-driven approaches [23, 26]. Indeed, automatic or semi-automatic code generation from Canonical Abstract Prototypes might be possible given formal definition of syntax and semantics. On the other hand, it is arguable whether this would be a desirable direction for further work. In particular, the final translation of a Canonical Abstract Prototype into an exact visual and interaction design is precisely where human problem-solving and creativity best come into play. Indeed, the abstract prototype precisely focuses the designer's attention on those issues of greatest importance and, through abstraction, invites innovation in interface design.

For highly conventional applications with modest design requirements and for which an uninspired, routine solution is sufficient—at least initially—automatically generated user interfaces might be acceptable. However, this is not the case for highly sophisticated and complex applications or if substantial improvements in human performance are sought. It is highly unlikely, for example, that any time in the foreseeable future an automated approach could lead to such breakthrough designs as the Siemens STEP-7 Lite [4, 6]. For such designs, skilled and inventive human designers are still needed, for which conceptual tools like Canonical Abstract Prototypes and processes like usage-centered design offer the necessary support.

References

1. Constantine, L. L., and Lockwood, L. A. D. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, Reading, MA, 1999.

2. Constantine, L. L., and Lockwood, L. A. D. "Usage-Centered Engineering for Web Applications." *IEEE Software*, 19(2), March/April. 2002.
3. Patton, J. "Extreme Design: Usage-Centered Design in XP and Agile Development. In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
4. Windl, H. "Designing a Winner: Creating STEP 7 Lite with Usage-Centered Design." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
5. Strobe, J. "Putting Usage-Centered Design to Work: Clinical Applications." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
6. Windl, H., and Constantine, L. "Performance-Centered Design: STEP 7 Lite." Winning submission, Performance-Centered Design 2001, <http://foruse.com/pcd/>
7. Constantine, L. L. "Essential Modeling: Use Cases for User Interfaces," *interactions* 2(2), March/April 1995.
8. Constantine, L. L., and lockwood, L. A. D. "Structure and Style in Use Cases for User Interfaces." In M. van Harmelan (ed.), *Object Modeling and User Interface Design*. Boston: Addison Wesley, 2001.
9. Constantine, L. L., Biddle, R., and Noble, J. "Usage-Centered Design and Software Engineering: Models for Integration." In *Proceedings, International Conference on Software Engineering, 2003*, Portland, OR, 3-9 May 2003.
10. Kruchten, P. *The Rational Unified Process: An Introduction*. Reading, MA: Addison-Wesley, 1999.
11. Jacobson, I., Booch, E. G., and Rumbaugh, J. *The Unified Software Development Process*. Reading, MA: Addison-Wesley, 1999.
12. Constantine, L. L. "Rapid Abstract Prototyping." *Software Development*, 6, (11), November 1998. Reprinted in S. Ambler and L. Constantine, eds., *The Unified Process Elaboration Phase: Best Practices in Implementing the UP*. CMP Books: Lawrence, KS, 2000.
13. Constantine, L. L., Windl, H., Noble, J., and Lockwood, L. A. D. "From Abstraction to Realization in User Interface Design: Abstract Prototypes Based on Canonical Components." Working Paper, The Convergence Colloquy, July 2000. www.foruse.com/articles/canonical.pdf
14. Constantine, L., and Henderson-Sellers, B. "Notation Matters. Part 1: Framing the Issues," *Report on Object Analysis and Design*, 2 (3): 25-29, September-October 1995.
15. Armstrong, C., and Underbakke, B. "Usage-Centered Design and the Rational Unified Process." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
16. Heumann, J. "Use Cases, Usability Requirements, and User Interfaces." Tutorial Notes, OOPSLA 2002, 4-8 November. ACM, New York, 2002.
17. Kruchten, P., Ahlqvist, S., and Byland., S. "User Interface Design in the Rational Unified Process." In M. van Harmelen, ed., *Object Modeling and User Interface Design*. Addison-Wesley, Boston, 2001.
18. Roberts, D., Berry, D., Isensee, S., and Mullaly, J. *Designing for the User with OVID: Bridging User Interface Design and Software Engineering*. Indianapolis, IN: Macmillan Technical Press, 1998.
19. Schneider, K. A., and Cordy, J. R. "Abstract User Interfaces: A Model and Notation to Support Plasticity in Interactive Systems." In C. Johnson (ed.), *DSV-IS 2001 Proceedings. LNCS 2220*. Berlin: Springer-Verlag, 2001, pp 28-49.
20. Volpert, W. "Work Design for Human Development." In C. Floyd, et al., eds., *Software Development and Reality Construction*. Berlin: Springer-Verlag, 1991.

21. Bodart, F., and Vanderdonckt, J. "On the Problem of Selecting Interaction Objects." In G. Cockton, S. W. Draper, G. R. S. Weir (eds.) *Proceedings of HCI'94: People and Computers IX*. Cambridge University Press, 1994, pp 163-178.
22. Pawson, R. R., and Mathews, R. *Naked Objects*. Chichester, England: Wiley, 2002.
23. Molina, P. J., Santiago, M., and Pastor, O. "User Interface Conceptual Design Patterns." In *Proceedings, DSV-IS 2002*. Rostock, Germany, June 2002: 201-214.
24. Constantine, L. L. "Abstract Design Patterns: Detail View Navigation." Working paper. <http://foruse.com/patterns/detailnavigation.pdf>
25. Garzotto F., P. Paolini, D. Bolchini, and S. Valenti,. "Modeling-by-patterns of Web Applications," in *Proc. Of the International workshop on the World-Wide Web and Conceptual Modelling , WWWCM'99*. Paris, November 1999: 293-306.
26. Molina, P. J., Belenguer, J., Pastor, O. "Describing Just-UI Concepts Using a Task Notation." In Joaquim Jorge, Nuno Nunes, and João Falcão e Cunha (eds.) *Proceedings of DSV - IS'2003 - 10th International Workshop on Design, Specification and Verification of Inter-active Systems*, LNCS - Lecture Notes in Computer Science. Berlin: Springer-Verlag.

Learn more about essential use cases and usage-centered design at
<http://www.forUse.com>.