Larry L. Constantine

Essential modeling is a systematic process for designing user interfaces that more simply and fully support what users are trying to accomplish. Essential use cases are abstract, generalized scenarios representing the essential patterns of interaction between users and a system. Use context models are abstract designs representing collections of capabilities and resources that a user interface must present to users in support of particular use cases. By keeping the focus on intent and purpose, essential models lead to user interfaces with improved usability built in from the start, thereby reducing the demands on usability testing, inspections, and iterative refinement.

Essential Modeling

Use Cases for User Interfaces

© FPG International

Practicing user interface designers and software developers need more than just rules and guidelines, they need practical tools and methods that point the way toward simpler and more supportive systems by highlighting the essentials of user intent and interaction. User interface design often focuses on the rules and the technology for the system interface itself, while techniques such as work-flow modeling and contextual inquiry [9] have concentrated on the work and tasks. What is needed is a bridge that helps practitioners directly connect the structure of the user interface to the structure of use.

Persistent questions are at the heart of this need. How do we distinguish what is truly necessary to support the work from what users may say they want or from what we are by habit prepared to offer? How can we describe and represent this work, and how can we then use our understanding of the work to design the architecture of the user interface to support it?

Essential use case modeling offers an approach to these questions that can be used within almost any user-centered design strategy. It was developed over several years by Lucy A. D. Lockwood and the author [6, 8] as a process that would be easy to learn and to apply, yet would dependably point toward user interfaces that fit markedly better with intended uses. This process is woven from two conceptual threads in systems analysis and design. The first thread is the concept of use cases, originally devised by Ivar Jacobson as a tool for object-oriented software engineering [10]. Use cases represent what a system offers to its users, the functionality of the system as viewed from the outside. When use cases are combined with the second thread, essential modeling, the result is a process that smoothly connects the design of user interface architecture back to the essential purposes of a system and the work it supports. This connection is achieved through three interdependent models: the user role model, the essential use case model, and the use context model.

Essential Modeling

Essential modeling is nothing new. Its roots trace back at least to structured design [15], where data flow diagrams were introduced for defining and describing application requirements apart from their implementation in software. Essential modeling eventually became a cornerstone of modern structured systems analysis [12].

Essential models aim to capture the essence of systems: a technology-free, idealized, and abstract picture grounded in the intentions of users and the fundamental purposes of the system that supports them. By assuming perfect technology—such as infinitely fast computers, arbitrarily large displays, keyless input from users, or whatever could most expeditiously realize necessary functions—models can be constructed that are free of unnecessary limitations or restrictive assumptions. The resulting design model can be more flexible, leaving open more options and accommodating changes in technology more readily.

In designing for usability, essential models can also serve another purpose. By identifying and representing the essential aspects of user requirements—the uses to which a system may be put and the interrelationships among these—user interfaces can be designed that more simply and straightforwardly meet basic user needs and better support the work users are trying to accomplish. Systems that support the work with fewer elements and features can actually be smaller and simpler to build.

Compared to models based in concrete behavior, such as task analyses or interaction scenarios, essential models highlight what it is that users are trying to accomplish and why they are doing it, in terms of their work and the larger context in which interactions are embedded. Essential use case modeling can be thought of as more usage-centered than user-centered, a "teleocentric" (purpose-centered) approach to design, rather than a "user-centric" one. Users are certainly not unimportant in this view, but they are most important as sources of understanding regarding the effective support of their work.

Use cases and essential use cases

Use cases are one particularly promising model of system usage. Sometimes referred to as scenarios [16], use cases represent the services or functions provided by a system expressed in ways that are meaningful to users. Each use case is one case of usage, a specific pattern or form of interaction that makes sense to a user. Use cases define system capability from an external or "black-box" perspective, but, as applied in software engineering, they deal with interactions not interfaces. Because they are written in terms of interactions with the features of a specified or assumed user interface, such use cases are not very helpful for designing the user interface; for this we need essential use cases.

An essential use case is a simplified and generalized form of use case, an abstract scenario for one complete and intrinsically useful interaction with a system as understood from the perspective of users who play a particular role in relation to the system. It is a description of the common or shared structure of an entire class or type of use to which a system may be put, expressed in simplified and generalized form. An essential use case contains the fewest presuppositions about technology, such as the shape or behavior of interface widgets, the layouts of screens or dialog boxes, or even the physical devices used for interaction.

To understand the difference between conventional, concrete use cases and essential use cases, consider the example of getting cash from an automatic teller machine. A concrete use case might take this form:

User inserts card; system reads magnetic stripe, requests PIN; user types PIN; system confirms PIN through EFT network, then prompts the user to select an action using the green buttons. The user keys in a selection, then is prompted to select an account type using the blue buttons; the user selects. The system prompts for an amount, the user keys in amount and presses the OK button to confirm. Finally the system tells the user to remove the card, the card is removed, and the system disgorges the desired cash, which the user removes and pockets.

Clearly, such a concrete scenario assumes not only features but detailed behavior of the user interface.

In forming an essential use case for this use, we first ask why the user, in the role of "bank customer," is using the machine in the first place. To the user, the purpose of this use case is to get cash; all else is interference or superfluous detail. As each step in the use case is considered, we again ask why. Why does the user insert a bank card? It serves to identify the user to the bank. Why does the user type in a PIN? It serves to verify that the current bankcard holder is really the authorized account holder, so that nobody else can take money from the user's account. Through simplification and generalization, we can construct the following essential use case:

Getting Cash

user action	*system response*
identify self	*verify identity*
	offer choices
select	*give money*
take money and run	

The essential use case leaves open more alternative ways for the system to offer choices to the user and for the user to make selections, including using new technology, such as touch-screens or voice response. It highlights that the ATM card and PIN are non-essential; their essential purpose is to identify and verify the user. Thumbprints, retinal scans, voice recognition, or badge readers might be workable and improved alternatives in some contexts. The essential model leaves open more possibilities, making it more likely that portions of our design will be reusable as assumptions and conditions change. It also points the way to simpler interfaces by highlighting the real heart of the matter to the user: getting the cash. Most users habitually take the same amount from the same account every time they use the machine. The first choice offered by the system could be this default case, user-defined or culled from past history: "$250.00 from regular checking, Mr. Chatworth?" Or whatever.

The essential model presents an ideal design target. A well-designed user interface would require only as many steps or as much information as expressed in the essential use case. The bank customer wants to be able to say, "It's me. The usual. Thanks!" and be off. We specify the ideal case because if we don't model it, we can't design to it. If we don't design to the ideal, we can't see where current practices or technical assumptions are limiting us, and we may miss completely the opportunity for better alternatives.

An essential use case has two parts — a user action model and a system response model — that describe the essence of what the user and

About the Author:
LARRY L. CONSTANTINE,
a consultant on the human and organizational aspects, is recognized for his pioneering work on structural design and for his widely used framework for fitting practices to organizational culture. Methods for improving software usability is a current focus of his research and teaching.

the system do in the course of the use case. The use case is expressed as a simple structured narrative in two columns, using the vocabulary of the end users and of the application domain. Use cases are given simple titles that convey ongoing action and essential purpose, for example: GettingCash, Searching, or SendingEmail.

User role model

Essential use cases are not invented from a vacuum but are derived from the roles that users play in relation to the system. For understanding usage, the roles that users play can be more important than the users themselves. A user role is an abstraction from the behavior of actual users who might interact with a system in similar or related ways. A user role is a collection of common interests, behaviors, and responsibilities [16] held by some class of users interacting with a system. Any number of actual users may function in a particular user role. Thus, for example, numerous bank employees may function in the role of "bank teller" in using an on-line bank transaction program. Conversely, any single user may function in a number of distinct roles in relationship to a system. For example, a bank supervisor may use the system in the role of "bank teller" when showing the system to a new employee or in the role of "supervisor" when making a correcting entry to override an erroneous transaction.

A user role model is simply a list of identified user roles along with their defining or distinguishing characteristics. Each role is named, briefly described, and characterized in terms appropriate to the design of the system. For example, a user of word processing software may operate in the role of writer or that of publisher. In the writer role, the user is primarily interested in getting ideas onto the screen and into saved document files, typically with little or no regard for appearance or layout. In this role, the user wants nothing to interrupt the flow of creative thought, nothing that might slow the efficient input and immediate editing of the words. In contrast, the user in the role of publisher, say of a company newsletter, is primarily interested in the layout and appearance of material on the page. To the extent that the content is important, interaction is likely to be concerned more with editing and changing than with originating material. However, the same person may sometimes function as writer, at other times as publisher.

The Essential Use Case Model

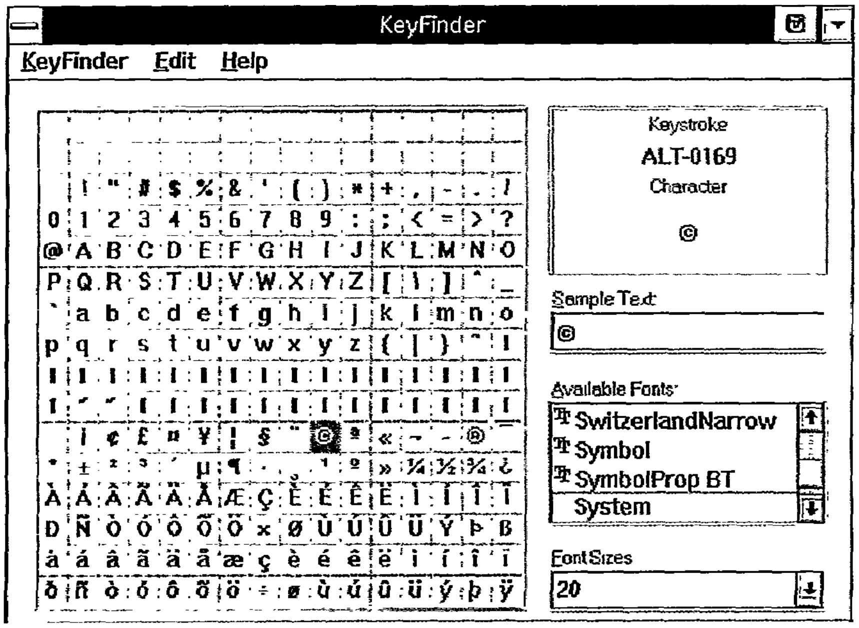
The ultimate goal in essential modeling is a coherent design for a complete user interface architecture. We want the components and organization of the user interface to fit closely the essential structure of the work being supported. For this reason, the essential use case model should reflect how use cases are interrelated as part of larger tasks.

Just as some tasks are more critical in a particular job, some use cases will be more central or salient than others. A complete use case model identifies one or more essential use cases as focal or primary, the usage around which an application is organized.

Use cases can be interrelated in numerous ways, among them affinity, classification, extension, and dependence. By making these relationships explicit, we can simplify the overall model and make it more fully and accurately represent the essential structure of the application or problem.

Affinity. Often, in the early stages of design, certain use cases will be seen to be more closely related than others, without the exact nature of that closeness or similarity being clear. An affinity diagram represents similarity or relationship of an unspecified form by the relative distance between visual elements in the diagram. Showing two things nearly on top of each other signifies that they are nearly identical; greater distances represent less "affinity," weaker interrelationship. The fact that affinity is imprecise and unrigorous makes it an appropriate construct for early stages of essential modeling in user interface design.

Classification. In some situations it may be clear that one use case is a subclass of another. For example, WithdrawingCash and Depositing are proper subclasses of an abstract use case UsingTellerMachine. This relationship is sometimes represented by the phrase "is-a-kind-of"



Keystroke
ALT-0169
Character
©

Sample Text
©

Available Fonts
SwitzerlandNarrow
Symbol
SymbolProp BT
System

Font Sizes
20

Figure 1
Example of a commercial keyboard extender applet (from Norton Desktop for Windows 3.0)

or, more simply, "is-a." Thus Depositing is-a-kind-of UsingTellerMachine. Indicating this relationship can simplify an essential use case model because subclasses to a common class have shared features or characteristics that can be described just once for the superclass, without being repeated for each subclass. Classification is used in essential use case modeling to simplify the model toward the end of simplifying the user interfaces designed from the model.

Extension. One use case may extend or alter the course of interaction of another. An extension represents inserted optional behavior, an exception, alternative, or special case. For example, in a graphics drawing application, Resizing could be a use case that may be inserted within the course of DrawingFigure. Keeping the optional extension separated from the use case being extended (DrawingFigure in this example), makes the narratives cleaner and simpler. Furthermore, the extension may be "reused" to extend any number of other use cases. Resizing might also extend InsertingClipart, but the extension need be described only once.

Dependence. One use case may depend on another use case as a subprocess or included interaction sequence. In a computer-aided software engineering tool, for example, labels may be a required element in several different graphical symbols. The essential use case model can identify a distinct use case, Labeling, that is used as a subprocess to the use cases for DefiningEntity and DefiningRelationship, for example.

Use context model

The use context model allows the designer to represent and manipulate the resources and capabilities that the user interface must present to the user. It provides an abstract and flexible model of the architecture of a user interface, leaving open details of appearance and choice of interface components. This kind of abstract model has also been called a work environment [9]. Unlike a paper prototype or design sketch, which is typically based on concrete visual features or user interface widgets, a use context model is a collection of abstract user interface elements representing needed or desired capabilities—materials, tools, and work areas—to be provided by a system in support of one or more use cases.

Use context models are easily constructed using a sheet of paper to represent each use context, with sticky notes for the materials, tools, and working areas. The sticky notes are readily rearranged, and their arbitrary shape and size is a reminder that they are abstract or representative features, not specific user interface widgets. Typically, these elements are identified in abstract or general terms that convey their functions, such as, "container," "work area," "selector," or "scratch pad."

Each use context is developed to support a specific use case or a set of closely related use cases based on affinity, classification, extension, and dependence as defined in the essential use case model. The goal is to avoid both unnecessary duplication and inappropriate combinations, always keeping the model as simple as possible while fully supporting all the use cases.

Essential Design: an Example

The process of design through essential modeling begins, of course, with users, but from there moves to the abstract roles they play in relation to the system being designed. The purpose of the user role model is to facilitate the identification of essential use cases, which are derived from a consideration of the needs, purposes, and intent of users in each of the identified user roles. Once identified, essential use cases can be organized into a coherent model of usage. Use context models are developed to support essential use cases or closely related groups of them. The use context models, in turn, guide the

General Typist *occasional use, single character, small symbol set*
Casual Translator *frequent use, 1-N character set, speed important; may operate keyboard from memory, need reminders, help with learning*
Casual Artist *try and retry; larger symbol set, appearance, position important*
Casual Technical Typist (math/science/engineering) *may retry; larger symbol set; appearance, position important*
Casual Coder *code conversion: ASCII, hex, DOS8, keyboard codes*

Figure 2
Map of essential roles for keyboard extender applet

derivation of a detailed user interface design, whether as a paper prototype or as software.

To understand this process in practice, a simplified example is helpful. This application is a deceptively small problem, rich with implications typical of real-world user interface design.

The keyboard extender problem

Frequently in the course of text or data entry from a keyboard, a user may need to enter a special character, such as a copyright sign or a "bullet" to flag an item in a list. Under Microsoft Windows it is possible to enter such characters using an "escape" sequence, such as Alt+0269. The user must remember the proper sequence of numbers and must enter them from the numeric keypad while holding down the Alt key. A visual "applet" to simplify the insertion of special characters is supplied with Windows itself as well as with various of its software enhancements. For example, with Norton Desktop for Windows comes the KeyFinder applet shown in Figure 1 (page 39). To use it, one must first switch to the applet or launch it if it is not already running. The desired character is selected by double clicking, causing it to appear in the "Sample text" box, where it must be highlighted, then copied to the Windows clipboard, either by typing Ctrl+C or selecting Copy from the Edit menu. Switching back to the original application, one merely pastes from the clipboard with Ctrl+V.

So much for ease of use! Seven or more distinct actions are required just to get a copyright sign into the text. Through essential modeling, it should be possible to design an equivalent applet with substantially improved usability.

Essential purpose of the keyboard extender

What is the essential purpose of such an applet? Why is it needed at all? Brief reflection suggests it is needed because not all desired characters are found on the keyboard. A major flaw is immediately apparent in the applets supplied with Windows and Norton Desktop: they present many characters that are actually on the keyboard, distracting the user, complicating the visual search task, and wasting screen real estate. Only the symbols not on the keyboard are of interest, because any symbol on the keyboard could simply be typed without recourse to the applet. The user is also only interested in a specific subset of special symbols: not some generic or system-defined set of symbols, but only those germane to the current task.

User roles in using the keyboard extender

We next identify user roles that imply differing needs and distinct patterns of usage. The situation introduced above represents a typical role that could be called "General Typist." This role is characterized by occasional use of the keyboard extender applet, usually to insert a single special character from among a small set of symbols, such as the copyright sign or a simple bullet. In contrast, the "Casual Translator" or multinational typist may have to type a phrase in Russian using the Cyrillic alphabet or a sentence in German making repeated use of ü (u-umlaut). We refer to this as the "casual" multinational typist, because a professional translator or serious typist in Germany or Russia would not be well served by any such applet. The Casual Translator role is characterized by more frequent or bunched usage, possibly involving an entire string of characters. For more intensive casual use, keyboard shortcuts are desirable so that the user can type special characters without activating or opening the applet. For this usage, the applet may be called upon as a reminder of how to input a particular character.

Among my acquaintances are those with a penchant for varied and fancy typefaces. (I myself have only 121 fonts installed on my laptop computer, although a somewhat greater number reside on my office system.) "Fancy font futzers," as they are sometimes called, like to play around with the appearance of documents, trying different typefaces in varied weights and sizes, experimenting with numerous "dingbats" or special shapes. They are likely to try and retry until they find just the right shaded box to mark the points in their memos. Appearance and position within a document are important to them. We can refer to this user role as the Casual Artist, to distinguish it from the serious graphic artist who would use a graphics drawing package or other professional software for their work.

Casual Technical Typists, with a need to enter an occasional equation or add engineering or scientific symbols to a report, have similar characteristics. A "real" mathematician or scientist would, of course, be more likely to use an equation editor or other specialized software. Finally we identify the Casual Coder, whose interest is the translation between the characters of ordinary discourse and their representation as codes in computers. In Figure 2 the major user roles and their characteristic activities are summarized. We need not be too concerned about whether we have exactly the "right" set of roles identified, as a certain amount of overlap or imprecision at this stage is not a major problem, and the model can always be refined later.

Essential use cases for the keyboard extender

Starting with the General Typist role, the first

InsertingSymbol

User Action	System Response
request	
	show symbols
select symbol	
	insert
	leave

TryingSymbol

User Action	System Response
request	
	show symbol
select symbol	
	insert it
[reposition/reshape it]	
repeat until satisfied	

InsertingStringl

User Action	System Response
request	
	show symbols
select symbol	
	add to string
repeat until all	
	insert string
	leave

DefiningGroup

User Action	System Response
request group	
	show group
[name/rename]	
select/deselect sumbols	
repeat until done	
	remember group

Browsing

User Action	System Response
request more	
	show more symbols
repeat until found	

Reminding

User Action	System Response
request	
	show symbols
do anything	
	leave

Figure 3
Essential use cases for keyboard extender applet

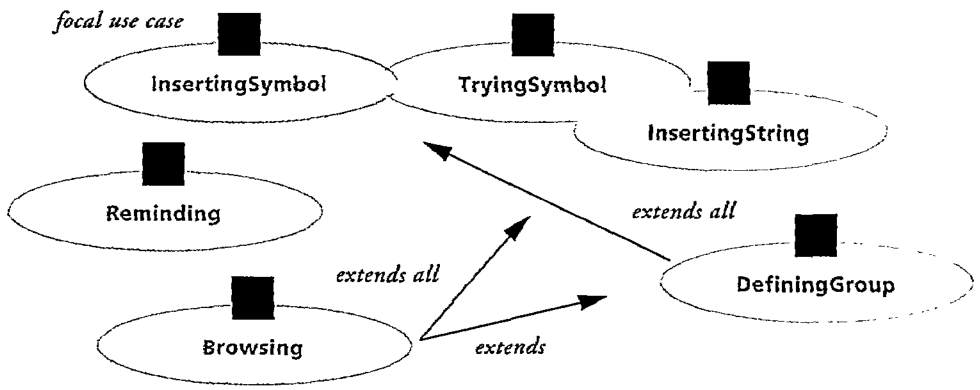


Figure 4
Essential use cases model for keyboard extender applet

essential use case we identify is InsertingSymbol. Sticking to essentials, the user must begin with a request (activating or switching to the applet), to which the system responds by showing candidate symbols, that is, a set of symbols of interest to the user. From this we infer the need for the user to communicate to the applet just which symbols are of interest. Rather than complicating the InsertingSymbol narrative with a digression on this optional alternative course of interaction, we identify DefiningGroup as a separate use case that extends InsertingSymbol and probably others as well.

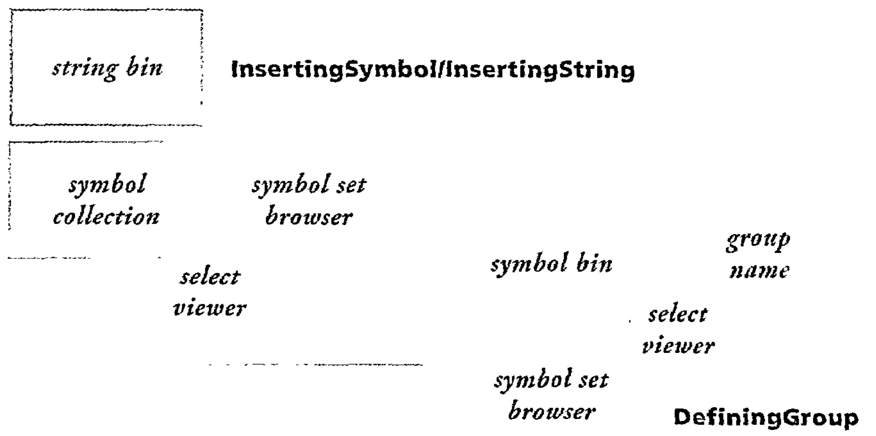
Returning to the focal use case, InsertingSymbol, the user must select the desired symbol, which the applet then inserts wherever the user was typing. Then the applet goes away. That is all. In designing to fit this essential use case our goal should be to require only two actions from the user—request applet and select symbol—not seven or eight.

But what if the desired symbol is not visible among those displayed by the applet? This suggests another use case, Browsing, which extends our focal use case. To browse, the user must ask to see more characters, and the system responds by showing more. This mini-dialogue continues until the user does something else, having found the desired character or given up. Note that the user seeks a particular symbol. The purpose is to browse through characters not fonts or type faces, which are, in many senses, artifacts of the data structure of the system.

From the Casual Translator role can be inferred the need for an InsertingString use case. This use case begins as with Inserting-Symbol, but a selected symbol is inserted into a string and the applet does not go away until the user explicitly finishes with all characters. Here, too, the user may not see the desired symbol among those offered, but we do not have to write another use case, since Browsing can also extend InsertingString.

Another use case, TryingSymbol, supports the Casual Artist or Casual Technical Typist roles. Here, the selected symbol is inserted, as in InsertingSymbol, but the applet remains until

Figure 5
Initial use contexts for selected use cases for keyboard extender applet



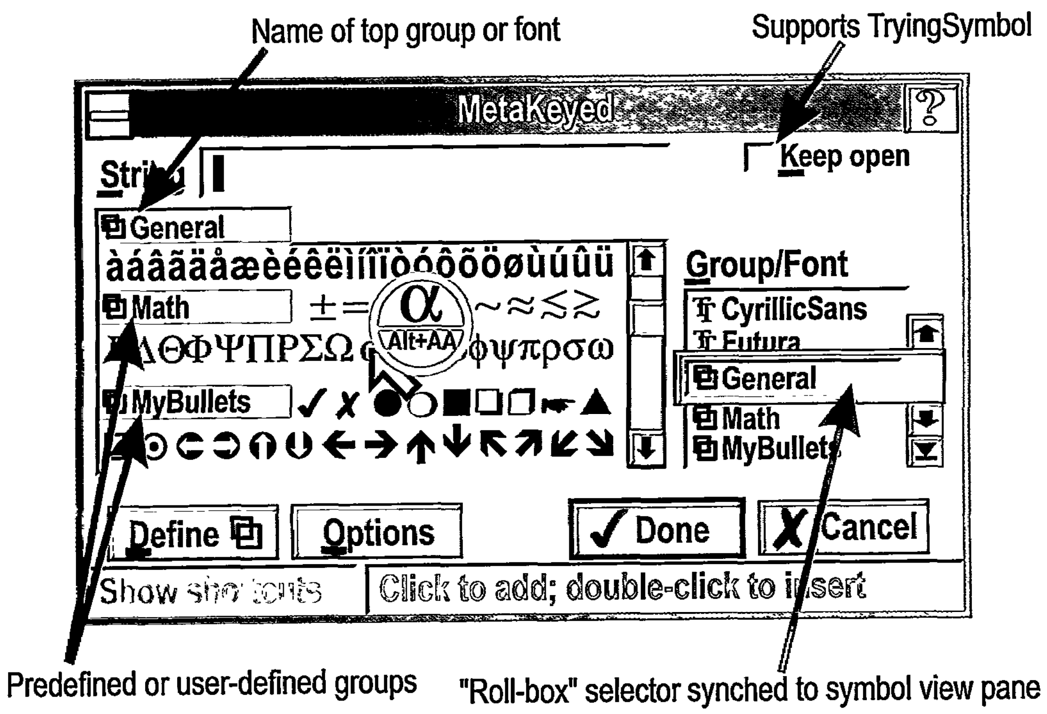


Figure 6
Initial paper prototype design for keyboard extender applet

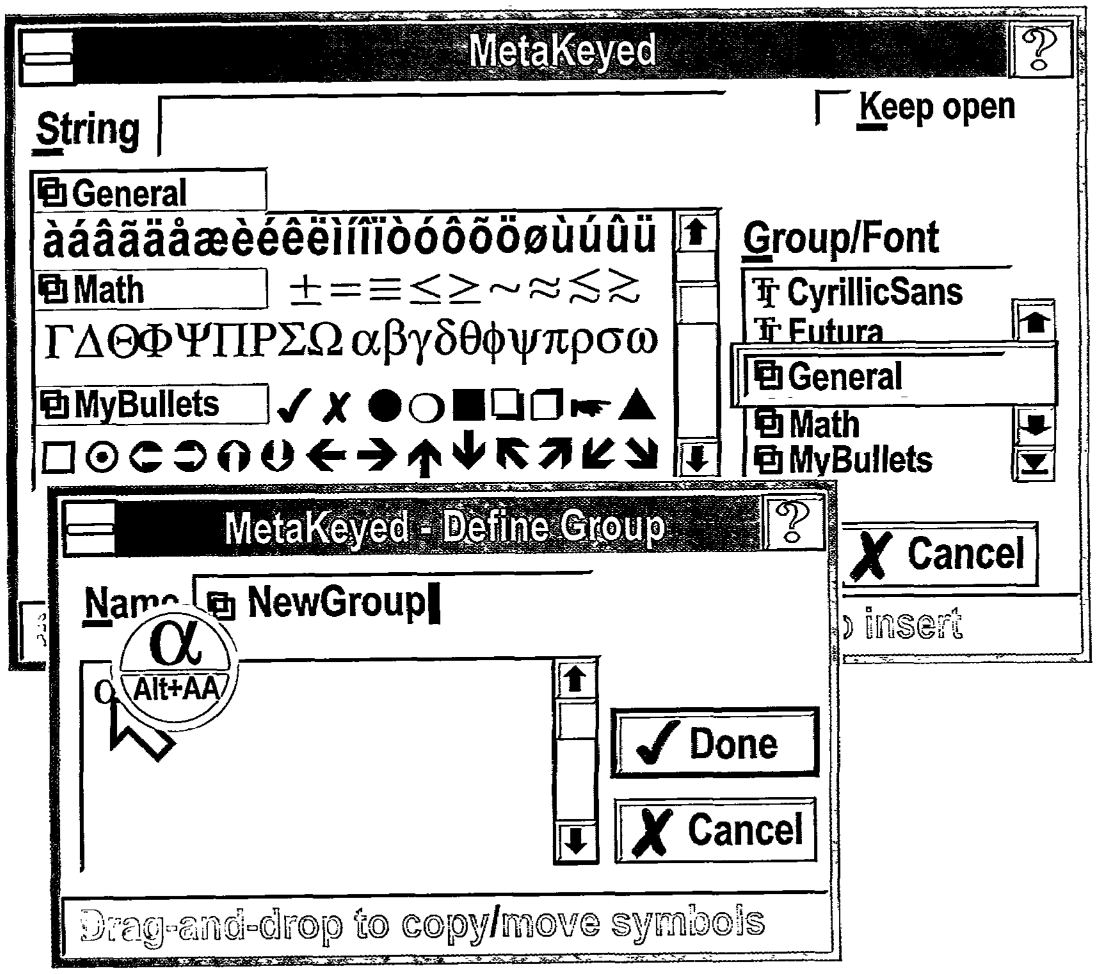


Figure 7
Initial prototype design for "Defining Group" essential use case

the user is finished, as with InsertingString.

The Casual Translator role implies the need to remind the user of the keyboard shortcut for a character. To the request for a reminder, the applet responds by showing symbols of possible interest along with their keyboarded equivalents. Any action by the user—selecting a character, typing something on the keyboard, clicking on some part of the document—that does not invoke another use case, such as Browsing, should make the applet go away.

Another essential use case identified earlier in the analysis is also needed. DefiningGroup allows users to define or redefine a group of symbols of interest for use much as if it were a type font or predefined symbol set.

For the sake of brevity, support for the Casual Coder will be skipped. The use case narratives are tabulated in Figure 3 (page 41), and a complete essential use case model is shown in Figure 4 (page 42). The close affinity among InsertingString, TryingSymbol, and InsertingSymbol is represented, with Reminding shown less closely related. Browsing is shown as extending all use cases; DefiningGroup extends everything else.

Larry L. Constantine,
Professor of Computing Sciences, University of Technology, Sydney
Principal Consultant, Constantine & Lockwood, Ltd.
8 Faneuil Hall Marketplace, Boston, MA 02109
72067.2631 @compuserve.com

From use context to user interface architecture

User interface architecture [3, 5] refers to the overall organization of the system interface as experienced by the user—not just features but their interrelationships, not just form but also functional behavior. The objective in the essential modeling process is to develop an interface architecture with an interconnected and compatible internal functional architecture that is optimized for identified uses as well as being capable of readily accommodating future elaborations and extensions. Although essential use case modeling can fit with any user interface and software construction method, it is especially suited to object-oriented programming, particularly object-oriented software engineering [10].

The design for the user interface architecture is derived directly from the use context models as informed by essential use cases. Use contexts are translated into specific interface composites—such as a screens, windows, panels, or dialogue boxes. Elements within use contexts are realized through user interface components that correspond as closely as possible to needed characteristics implied by the essential models. In many cases improved usability results from first "inventing" whatever custom controls best fit the context, then selecting best-fit compromises from among available standard controls, such as, list boxes, radio buttons, or text entry areas.

Not infrequently, however, the essential models imply that standard user interface components and practices actually fail to support usability goals. Such is the case, for example, with standard scroll bars as devices for visual navigational through graphical spaces [4]. Departures from standard look-and-feel must always be considered carefully, but consistency itself may be an overrated commodity, especially when consistency runs counter to the best support of user work [14]. Where there is no near equivalent among standard widgets, actual design and implementation of custom controls may be the best option, especially considering that this is becoming relatively easy in new object-oriented visual development environments, such as Borland's Delphi and IBM's Visual Age.

The derivation of a prototype design is clearly not a mechanical translation but a creative design process. Based on experience and the application of established usability principles [2], the designer may choose to combine several use contexts into one interface composite or decide to split one use context across two or more dialogues or windows. However, the choice, design, and layout of the specific components in the prototype design is always guided by the essential use case model and the use context model.

Figure 5 (page 42) illustrates two of the use contexts that might be derived from the essential use case model. Note that some use cases have been combined based on their similarity or other relationships.

In deriving an initial design for a prototype of the keyboard extender, we are guided by everything learned in the process of constructing the essential models. We know that users are interested in specific special symbols defined by their task but not found on the keyboard. Their interest is firstly in symbols, not in fonts or typefaces, which are for many purposes just artifacts of the internal data structure. This means that the display should organize characters in meaningful pre-defined and user-defined groups independent of fonts and allow users to browse characters without necessarily selecting by such groups or fonts, which are easily forgotten or confused.

A somewhat idealized initial prototype to cover selected use cases is shown in Figure 6 (page 43). The hint on the status bar indicates how this design implements the competing requirements of the InsertingSymbol and InsertingString use cases. A double-click on a symbol in the browsing box inserts it in the users text and dismisses the applet; a single click appends it to the string in the text box above. The string can be inserted by clicking on the "Done" button or by double-clicking the last selection. Because the string is built in a text box and the keyboard is active, mixtures of selected special characters with regular ones from the keyboard are easily constructed, thus supporting the Casual Translator role. This prototype allows the Casual Artist or other user to set a check-box toggle to keep the applet active and visible. Alternative mechanisms to support TryingSymbol without impeding the focal use cases may have to be evaluated through objective testing or usability inspections.[13]

A custom scrolling control has been designed to allow visual browsing through collections of symbols that may derive from multiple fonts or user-defined groups. Because users may sometimes want to know the names of the groups or fonts from which symbols are selected, these are carried with the collections in the browsing box. The name of the top group or font "sticks" at the top line rather than scrolling off, so the user always sees the name with the corresponding collection of characters. Where selection by font or group is desired by the user, a scrolling list box or drop-down list box would work, but in this prototype, a "roll-box" has been specified that keeps the selected font or group centered in the visual field. The character browsing panel is synchronized to the roll-box so that rapid visual browsing of symbols through multiple group/fonts is easy even when selection is made through the font/group roll-box. Contrast this to the multistep scenario required using a standard drop-down pick list: click to drop list, find font/group, click to select, look for symbol, click to drop list again, etc. For those users who prefer this familiar behavior, the standard control is also provided.

Figure 7 (page 43) shows prototype support for the DefiningGroup use case. An attached panel opens allowing groups to be defined or redefined simply by familiar drag-and-drop operations from the already available symbol and font browsers above. In this way customization is achieved without requiring users to learn any new skills or behaviors.[4, 6] The interface is also more visually stable than if browsing and viewing facilities were replicated in a separate "customization" dialogue.

Other features of this tentative design include a magnifying character viewer so that it is easier to distinguish subtle differences in small symbols, e.g., í (i-acute-accent) and ì (i-grave-accent). The viewer also reminds the interested user—at the point of visual focus—of the keyboard shortcut for direct entry of the symbol.

This prototype, a composite of student solutions designed in training workshops, is neither complete nor perfect. It is presented here simply to illustrate the next step in the essential design process. A final version would emerge through successive refinements, inspections, and usability testing—already familiar processes to the practicing usability specialist.

Essential Conclusions

Essential use case modeling is neither a methodology for user interface design nor a substitute for user-centered approaches, but a process that augments and facilitates user-centered approaches. Essential use cases are particularly effective as a medium for communicating with users about their work and the system requirements to support that work. Users do not need to learn a new language, an obscure notation, or a specialized set of modeling conventions to understand essential use case narratives. These are expressed in the language of the user and are, if reasonably well-constructed, completely self-explanatory. Use cases can even be used to help redesign and simplify the work itself and have been applied to this end in business process reengineering [11].

By identifying and designing to the ideal case, unconstrained by physical or practical limitations, designers are able to free up their thinking to devise new and more creative solutions to user interface problems. Through an understanding of these abstract and idealized cases, designers can recognize where current software and hardware technology are limiting factors and where they are not. In capturing uses relatively independently of technology and representational aspects, essential modeling leaves open more options in the detailed design of the user interface. By tying the details of the user interface and their interrelationships more closely to the underlying needs of users it becomes possible to build smaller, simpler systems that fully support the work of users. Finally, by specifying an ideal and optimal target for design, user interface designers will have a better chance of offering users what they really need.

Acknowledgement
The support of ISSC Australia, a joint-venture of IBM and Lend Lease, is gratefully acknowledged.

References

[1] Constantine, L. L. Complexity and creeping featurism. *Computer Language Magazine* 9, 10 (October 1992).

[2] Constantine, L. L. Getting the User Interface Right: Basic Principles. *Software Development '93 Conference Proceedings.* Miller Freeman, San Francisco, 1993.

[3] Constantine, L. L. Interfaces for intermediates. *IEEE Software* 11, 4 (July 1994): 96–99.

[4] Constantine, L. L. Graphical navigation. *Windows Tech Journal* 3, 8 (August 1994).

[5] Constantine, L. L. "Persistent Usability: A Multiphasic User Interface Architecture for Supporting the Full Usage Lifecycle," in S. Howard and Y. K. Leung, eds., *OzCHI 94 Proceedings.* Melbourne, November 1994.

[6] Constantine, L. L. Essentially speaking. *Software Development* 2, 11 (November 1994): 95–96.

[7] Constantine, L. L. *Constantine on Peopleware.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[8] Constantine, L. L., and Lockwood, L. A. D. L. *Essential Use Cases: Essential Modeling for User Interface Design.* Tutorial Notes, OzCHI 94, Melbourne, November 1994.

[9] Holtzblatt, K., and Beyer, H. Making customer-centered design work for teams. *Communications of the ACM* 36, 10 (October 1993).

[10] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, Reading, Mass., 1992.

[11] Jacobson, I., Ericsson, M., and Jacobson, A. *The Object Advantage: Business Process Reengineering with Object Technology.* Addison-Wesley, Reading, Mass., 1994.

[12] McMenamin, S., and Palmer, J. *Essential Systems Analysis.* Prentice Hall, Englewood Cliffs, N.J., 1984.

[13] Nielsen, J. *Usability Engineering.* Boston: Academic Press, 1993.

[14] Raskin, J. Intuitive equals familiar. *Communications of the ACM* 37, 9 (September 1994): 17–18.

[15] Stevens, W. P., Myers, G. J., and Constantine, L. L. Structured design. *IBM Systems Journal* 13, 2 (May 1974).

[16] Wirfs-Brock, R. Designing scenarios: making the case for a use case framework. *Smalltalk Report,* November-December 1993.