An Approach to Specifying Requirements for User Interfaces

Grady H. Campbell, Jr.

Spectrum, developed from 1984–1988 at Template Software, was an application engineering environment designed to support the construction of workstation–based information and decision support systems. Spectrum users constructed systems by describing the properties of a required system in Spectrum's notation. This description was in principle a requirements specification (or application model), formalized sufficiently to drive automatic derivation of a standardized design and selection, adaptation, and composition of reusable components producing a corresponding system. A specification for an application system consisted primarily of:

- A “world model" represented as an active semantic data model consisting of classes of objects having associated attributes and computations.
- An “external interface" described as a set of “logical devices" having associated inputs and outputs.

The concept of a Spectrum specification was modeled on the principles of the SCR requirements specification [1]. The external interface concept was also influenced by research in user interfaces (e.g., [2], [3], [4], [5]). The world model concept was conceived as a formalization of the term dictionary of the SCR requirements, influenced by research in semantic data models (surveyed in [6]) and needed to enable automatic software generation. The purpose of this paper is to describe the approach to specifying a user interface exemplified in Spectrum and to compare this with other popular approaches to creating a user interface.

The Spectrum Model of User Interface Devices and Input/Output

For information and decision support systems, a primary type of device that should be specifiable is a “user display." A user display corresponds to a logical user role in an envisioned system. The Spectrum model of user interfaces comprised a variety of concepts. The following items informally characterize this model:

- **User Interface Device.** The basic physical user interface device, consisting of both hardware and software, is a general–purpose, windowing CRT screen with keyboard and mouse. (Variations on this type of device and other types of user interface device, such as printers, are treated similarly but not discussed here.) The physical device supports multiple logical user interface devices, each corresponding to a user role. Each logical device is independently mapped onto the physical device, abstracting that device's mechanisms and sharing its resources with other logical devices. (The TC–2 Panel is the closest A–7 analog to this type of device.)
- **User Interaction.** User interaction with a system occurs via a logical user interface device associated with a particular user role. That device is characterized by a set of output displays. The displays are organized into a control structure in which active displays exhibit data, enable input of data, enable activation of computational processes (defined by the world model), and enable activation of other output displays.
- **Outputs**. The world model defines the data represented in the system. An output is a displayable representation of some portion of that data, displayed either in a window of the logical device or as a component of a larger output. Displayable

representations include text, images (such as icons, maps, and pictures), graphics, audio, video, and compositions of these (such as text labeling a picture or a path on a map). Each representation has associated attributes, such as color and font for text or volume for audio. A data item can be simple or arbitrarily composite. A composite data item may be displayed in a monolithic form that abstracts its content or in a composite form having parts analogous to and derived from its internal structure. Outputs have physical dimensions and are mapped into the coordinate space of a logical user interface device screen.

- **Inputs.** Just as outputs are representations of internal data, inputs are user expressions of data to be represented internally. Keyboard and mouse buttons are basic mechanisms of input. Beyond these basic mechanisms, extended input mechanisms can be represented in abstract forms combining basic input and output mechanisms as needed to represent the type of data being input. Extended mechanisms include text entry, selection of items from a menu (which can be inferred from actions enabled within an output context), and graphical entry mechanisms such as toggles, switches, pushbuttons, sliders, and dials.
- **Input/Output Correlation.** User inputs are very context sensitive. They are usually defined/enabled within the context (physical boundaries) of a particular output. A particular input can mean different things when received within the `boundaries' of different outputs (i.e., there is a concept of the `location' of an input relative to active outputs). Within a composite output, an input can mean different things at different locations within the output. Exceptions to this correlation of input to output, analogous to sensor device polling, is the initiation (prompting) of user input as a world model source of information, as part of context setting for user interaction, and as ancillary data which is part of a composite input.
- **User Interface/World Model Correlation.** A user interface displays the information `known' to the system as defined by the world model. Although there can be different ways to display particular information, the form of the information can narrow the developer's choice to feasible alternatives or engender a preliminary or default view without additional developer input. Because outputs are conceived as views into a conceptual database defined by the world model, access to data is implied by the specification of outputs. Because access is an implicit mechanism, redisplay of data when changes occur is implicit. Similarly, associated world model mechanisms (such as add, change, copy, and delete of data) imply capabilities for input action.

The Spectrum User Interface Concept

The Spectrum capability to specify (and generate) a user interface was an implementation of the model of user interfaces described above. A user interface is a set of user roles. Each user role is defined by an organized set of displays. A display is defined by its information context (a class in the world model), its content (a simple or composite output), its format, and associated actions (including inputs and activation of related outputs). The displays that make up a user role are organized into a control flow that is realized dynamically based on viewed information and associated user actions.

The following is a simplified (and somewhat incomplete) view of the abstract structure of a Spectrum user interface specification ({...} indicates repetition, [... | ... | ...] indicates alternatives, "..." is an elaborating comment):

User interface ::= {Display}	"an interface is a set of displays, each defining a user role"
Display ::= Context	"the objects viewed through this display"
[Controller | Output]	"specification of a controller or output"
{Input}	"inputs that are available across the entire display"
{Display}	"Displays which are accessible within the containing display"
Controller ::= {Display}	"the set of displays managed by the controller"
Sequencing	"the control regime followed by this controller"
Sequencing ::= [sequential	"indicates a sequence of outputs/controllers"
| selection	"indicates a menu of outputs/controllers"
| multiple]	"indicates multiple concurrent outputs/controllers"
Context ::= Context_class	"a candidate set of objects to be displayed"
selector	"a predicate to filter the object set"
Output ::= [Form | hierarchy | document | graphic]	"(this is a conjectured, extensible set of output types; only Form is expanded here; all output types would be expanded analogously)"
Form ::= {Field}	"views into attributes of each context object"
Field ::= precondition	"a predicate for whether a field is visible based on the referenced object (e.g., in a specified subclass)"
[Value	"concrete data (alphanumeric, image, audio, video)"
| Relation	"a related set of objects that provide a new context"
| Display]	"an embedded display of a context object"
format	"constrains where the field is and how it looks"
{Input}	"inputs associated with this specific field only"
Relation ::= selector	"a predicate to filter the related object set"
Display	"a nested display of related objects"
Input ::= precondition	"a predicate for whether the action is enabled"
action	"a triggering key and/or menu entry descriptor"
effect	"meaning of the action, including input of ancillary data and world model update and process activation"

A Spectrum user interface specification entails substantial implicit semantics. Implicit semantics can be a source of expressive power or an imposition of inflexibility. An effective approach to specifying user interfaces must find a balance between inflexibility and requiring expression of too much incidental detail. A partial solution is to assume defaults at a particular level of

specification but allow the developer to work at lower levels to refine the details of a specification and supersede defaults. Alternatively, when it is possible to establish standards or conventions, the developer is relieved of describing the corresponding details but can assume that the user interface will be implemented in keeping with the implications of those standards and conventions. In Spectrum, the specification of a user interface was simplified by limiting output descriptions to a stylized set of formats and by defaulting the format of each output based on corresponding world model definitions.

Comparisons with Other Approaches

Approaches to specifying a user interface can be compared by considering whether each is prescriptive or descriptive, where the concepts it supports lie on the continuum from hardware mechanism (implementation) to abstract interaction (requirements), and how much a specification contributes to the creation of a complete system. As a rule, the ideal approach is one that is descriptive without unreasonable limitations on developer flexibility, represents interfaces in abstract terms matching the concepts of interaction natural to the problem being solved, and is an integral element of a complete system specification. The Spectrum approach, on this criteria, was properly descriptive and an integral element of a complete specification but was not sufficiently flexible in its concepts of representation and interaction. Other approaches to user interface specification are:

- **User interface toolkits.** Compared with a descriptive specification of a user interface, a toolkit supports a prescriptive specification of a user interface which can give the builder more flexibility in creating exactly the right interface. Examples of toolkits are the various X window system toolkits, Microsoft's Windows SDK, and Apple's Macintosh Toolbox. If a notation for descriptive specification does not give the builder the right concepts or options, creating the desired interface can be impossible or require compromises. However, the toolkit approach tends to blur the distinction between requirements and engineering decisions. In addition, the cost in using a toolkit is that the builder has to deal with all of the details of the interface, including many that could be standardized by convention or inferred from the specification of a particular interface.
- **Visual (direct manipulation) user interface builders.** The attraction of a visual interface builder is that it is both descriptive and the developer can directly control the way the interface will look. Examples of tools that include visual interface builders are NextStep (NeXT Computer Inc.), Open Interface (Neuron Data Inc.), Galaxy Application Environment (Visix Software Inc.), and XVT (XVT Software Inc.). Using either a toolkit or a descriptive form fails to provide the builder with as concrete a sense of what the interface really looks like. Such a sense can be gained indirectly by generating the software and running it, or there may even be a tool that can interpret the description and present the results to the builder. However, unless the developer can manipulate the result to refine its looks, getting exactly the intended look can be difficult and require repeated changes to the program or interface description. Similar to the toolkit approach, this approach mixes requirements and engineering decisions in a single representation. The visual approach is also limiting if it forces the builder to rely on static positioning or appearance when variable positioning or appearance would result in a more effective interface.

- **Domain–specific user interface specifications.** Spectrum interface concepts are domain–independent. The Synthesis methodology for reuse–driven software processes (defined in [7]) takes the view that, within a particular domain, there may be standards for formatting or terminology that need not be reinvented, or even explicitly stated, when specifying a user interface for a system in that domain. In other words, the specification language can be tailored to the needs of a particular domain, making the language more expressive and therefore reducing the effort to describe an interface that adheres to the prescribed standards. Another way to think of this is that, within a domain, there is more commonality in constructible systems' user interfaces and less variability need be expressible.

A Related Issue: End–User Interface Tailoring

Many of the details of a user interface can be determined at any of three times: while the software is being developed, while the software is being installed, or while the software is being used. The latter two must be enabled by adding corresponding functionality to the software as it is developed. Although resolving those details in software development tends to result in the most efficient software and simplest interface, the alternatives are necessary when the details cannot be constrained at development time because there are different users with different needs or preferences that must be satisfied. However, preferences are often just that and the required additional complexity in installation or system use does not justify this flexibility in the user interface. The easy answer for many systems seems to be to just build all applications with a plethora of user interface tailoring mechanisms that have nothing to do with the intended purpose of the system, without regard to the complexity it adds.

Future Directions in User Interfaces

The trend for computers and user interfaces is toward integration of all types of media and interaction. Effective use of audio and video, collaborative groupwork, and distributed computing are just beginning to be understood and provide a substantial challenge in creating user interfaces. Laurel [8] describes a compelling paradigm for user interfaces, originating in work on computer games and based on an analogy with theater in which the audience both observes the action and interacts with the actors (which in this case can be either human or software). Emerging ideas in visualization and virtual reality may have similar implications for the user interface concept. Specifying the sort of interface concept that results will require a much more powerful and comprehensive concept of the nature and mechanisms of interaction.

References

1. Kathryn Heninger, John Kallander, John Shore, and David Parnas, *Software Requirements for the A–7E Aircraft*, NRL Memo. Report 3876, Naval Research Lab., Washington, D.C., November 1978.
2. Mary Shaw, et al. “Descartes: A Programming-Language Approach to Interactive Display Interfaces," *Proceedings of the SIGPLAN '83 Symposium on Programming Language Issues in Software Systems* (ACM SIGPLAN Notices 18(6)), June 1983, 100-111.
3. Lawrence Rowe and Kurt Shoens, “Programming Language Constructs for Screen Definition," *IEEE Trans. on Software Engineering* SE–9 (1), January 1983, 31–39.

4. *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation* (ACM SIGPLAN Notices 16 (6)), June 1981.
5. Richard E. Fikes, “Odyssey: A Knowledge-Based Assistant," *Artificial Intelligence* 16 (3), July 1981, 331-361.
6. Alexander Borgida, “Features of Languages for the Development of Information Systems at the Conceptual Level," *IEEE Software* 2 (1), January 1985, 63–72.
7. *Reuse–driven Software Processes Guidebook*, SPC–92019–CMC, v. 02.00.03, Software Productivity Consortium, Herndon, VA., November 1993.
8. Brenda Laurel, *Computers as Theater*., Addison–Wesley Publishing Co., Reading, MA., 1992.

Specifying Requirements for User Interfaces

November 30, 1994

Grady Campbell

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

SOFTWARE
PRODUCTIVITY
CONSORTIUM

Principles of SCR Requirements Applied to User Interfaces

- Describe externally visible behavior: what users can see and do, from the system's perspective
- Organize to separate concerns: interface device capabilities, interaction control, logical outputs and inputs
- Use formally defined notation: templates derived from a framework of abstract U.I. specification concepts

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

Approaches to U.I. Requirements Specification

- Ad hoc descriptive (prose/pictorial)
- Prototyping
  - Constructive (U.I. toolkits)
  - Direct manipulation (visual U.I. builders)
- Semi-Formal
  - Generic framework (e.g., Spectrum)
  - Domain-specific framework (e.g., Synthesis)
- Formal

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

Spectrum Specification Framework

World Model (Active semantic data model: Object classes, attributes, computations)

External Interface

- User Interface Devices (hardware and software)
- User Interaction (Control): Logical Devices or Roles
- Output Displays
- Inputs
- Input/Output Correlation
- User Interface/World Model Correlation

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

Partial U.I. Specification Concepts

User Interface := {Display}

Display := precondition Context
[Controller | Output]
{Input}

Controller := {Display} Sequencing

Sequencing := [sequential
| selection | multiple]

Context := context_class selector

Output := [Form | hierarchy
| document | graphic]

Form := {Field}

Field := precondition
[value | Relation
| Display]
format
{Input}

Relation := selector Display

Input := precondition
action
[effect | Display]

Implicit semantics: data access mechanisms, dependency-driven display updates, default inputs, logical/physical device mapping, internal/external data transformations, layout implied by different output types, concept of a user 'session'

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

U.I. Specification Features

- A user interface organizes user/system interaction into roles.
- A display organizes a collection of outputs into a structured dialogue.
- Each output, which is (only) a logical view into a world model, can be arbitrarily complex (composed of multiple data values and other outputs).
- Each input is defined relative to an output or output component.
- Every output and input may be conditioned on the content of referenced objects.

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

A Framework for Formal Specification?

- Formal specification of a user interface would be extremely complex and excessively detailed.
- U.I. framework approach is analogous to programming languages approach: pragmatic and loosely based on formal semantics of primitives and abstract machine
- Formal U.I. framework definition + semi-formal U.I. specification may enable derivation of formal U.I. specification

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.

Some Issues

- Integration of U.I. specification within a system specification
- 'Complete' U.I. specification concept framework
- End-user tailoring capabilities (install-time or run-time specification)
- Accommodation of emerging interface paradigms

Copyright © 1994, Software Productivity Consortium, Inc. All rights reserved.