Tailoring Mechanisms in Three Research Technologies.

Austin Henderson
Rivendel Consulting
PO Box 334
La Honda, CA 94020 USA
+1 650-747-9201
henderson@rivcons.com

ABSTRACT
Tailoring is the technical and human art of modifying the functionality of technology while the technology is in use in the field. This position paper explores various styles of, and mechanisms for, tailoring in three research systems (Trillium, Rooms, and Buttons) created by the author to explore ways to enable players (end users) to achieve new behaviors from these systems appropriate to their particular circumstances.

Keywords
Tailoring, changing functionality, multiple truths, Trillium, Rooms, Buttons.

TAILORING
Technology in use by a player doing some activity must deliver functionality appropriate to the circumstances of that use. The player must map the funcitonality of the technology into the activity, determining what parts of the activity will to represent in the technology, and how. Technologies are designed to make such mappings easy for particular target activities.

Since a technology can be used in different activities, the stretch of the mapping can be as wide as a person wishes to make it. However, if there is regularity to the stretched mapping, it is common for the player to want to adjust the functionality of the technology to make the mapping easier, and thereby to let the technology do more work in the activity.

Sometimes these changes can be made at design time, when the technology is being developed. But when this is not possible, the functionality of the technology needs to change when the technology is already in use in the field. Tailoring is the technical and human art of making such changes [1].

This position paper recalls and analyzes experiences with three research systems, Trillium, Rooms, and Buttons that were built with the need for tailoring as an explicit focus.

TRILLIUM
Trillium [2,3,4] is a user interface design environment used in the past 15 years at Xerox corporation. Trillium asks the player (a designer of user interfaces) to model four things: the machine being presented, the navigational structure of the interface, the style of the interface, and the interactional screens. The machine is modeled by a set of state variables (cells) whose values were set by the user interface both to convey parameters and invoke action. Navigation is modeled as stack-based movement between compositionally defined frames. The user interface style is modeled out of collections of objects which are instances of parametrized types. Types are the language of the design for a particular interactive style and are created compositionally over a primitive set that capture both the behavior and presentational aspects of the interface.

Trillium was tailorable in a number of ways: Its design anticipated various different ways that players will want to have Trillium behave in enabling design activity, providing a large array of preference settings. Almost all designers made use of these mechanisms.

Scaffolding
The targeted activity for Trillium was design: a designer using Trillium was creating items which together would simulate/implement a user interface. Thus even the target activity involved changing the behavior of a created object. In doing so, the designer could just use Trillium as provided without change, something that would not on first encounter seem to involve any tailoring.

However, it was not uncommon for designers to add items to their developing user interface (the design target), items that helped to get that interface into configurations suitable for continued design. For example, although in its final form the interface would have a carefully planned way of exposing more detailed choices (progressive disclosure), designers would often have a button in the opening frame that would jump directly to the frame where design was currently proceeding. On resuming work, the designer would set the interface running by switching Trillium into "run" mode, push this special button to get to where they should continue work, and then switch into design mode.

Thus since Trillium was a design environment for things which had behavior, behavior constructed as part of the target object (Trillium's subject matter) could be harnessed to augment the behavior of the design environment (Trillium's inherent behavior). Designers referred to such "temporary" additions to the target design as "scaffolding". Scaffolding would in the end be removed, but before that—sometimes for a very extended time—it was be an integral part of the designed object. Yet its purpose was for tailoring—extending Trillium's behavior as a design environment. Further, inevitably and intentionally, scaffolding extensions were specific to the design being carried out, defined and created in the context of that object.

Scaffolding is a very common form of tailoring, a boundary-crossing behavior between two related design environments—the target design, and the tools design. People do this in almost every design environment. In graphics and engineering design environments, people draw construction lines, insert objects as tools. Where the target objects have behavior, scaffolding can be used to extend the behavior of the desgin environment. The best of this is where the design enviroment has anticipated this, and provided for such target objects designed as scaffolding to extend the design environment. This requires that the design environment and the target environment be compatible, so that target objects fit well into the design environment. Programming environments, particularly interactive and interpretive ones, are well-know for this (e.g., Lisp, Hypercard, and the programmable text editor Emacs). [I regard integrated scaffolding as essential for quality programming environments.]

Construction set: pieces and glue

Trillium provided a construction set that enable the creation of an open-ended set of itemtypes. An itemtype was specified as a description (the glue) in a language specially created for the purpose of indicating the parameters of the new itemtype and how, given values for the parameters of an item, the item was to be understood, and implemented, as a set of component items (the pieces). When a new set of interfaces was imagined (e.g., the metaphor of filefolder tabs as a segmentation mechanism in selections among many choices), new itemtypes that expressed that metaphor (e.g. tabs, sets of selection choices) were created. This creation effectively tailored Trillium to the activity of creating these new sorts of user interfaces.

Extensibility

The primitive itemtypes of this construction set (the smallest pieces) anticipated the set of interfaces that could be built by expressing the base set of mechanisms for presentation (artwork, sensors, and displayers), and interaction modeling (initializers, implication, inhibitors). This base set for the construction set was extensible by providing an API in the language of Trillium's implementation (Interlisp-D) for each class of mechanisms.

The creation of primitive itemtypes was rarely the province of designers, although designers did engage in the design of such new itemtypes. Mostly this was lef to the "supporters", design-literate programmers who were essential to the larger-scale success of Trillium at any site.

Over the years of Trillium's use, only a few designers chose to learn how to create new primitive items, and these were designers who already knew how to program. Still, as the demands of this sort of programming within the Trillium framework was quite confined, and as new itemtypes were typically created by copying and editing existing ones, and as the editors used for editing Interlisp code were the same ones as were used for editing Trillium structures, the more experienced a designer had become with Trillium, the more likely they were to have at least attempted to play supporter and create new primitive itemtypes.

Brain surgery

The behavior of the Trillium's primitive itemtypes were understood in the context of "the Trillium interpreter", a virtual machine that made calls to the functions of the API's of the primitive items. It also provided the navigation "stack" that recorded the frames which had been previously visited, and the inheritance structure permitting frames to be composed of other (usually shared) "subframes". This interpreter was hardwired, and was formally changed on rarely, and then only when the community of designers and supporters agreed on a better way of thinking about the basic structure of Trillium.

However, informally, improvements were explored by many in the community, with the supporters modifying the "hardwaired" implementation. Again, this was enabled by Interlisp-D (like most Lisp environments) being openly augmentable and advisable. Nothing in Trillium made such adjustments easier or harder than any other application written in this environment. However, the social and organizational practices surrounding such changes were critical to the success of this experimentation. They, and nothing in the technoogy, insured that experiments were identified as such, and remained outside produiction environments until approved by the Trillium community as a whole.

ROOMS

Rooms [5,6] was an extension to the Interlisp-D application environment that provided multiple virtual desktops, each supporting an assemblage and configuration of documents appropriate for a particular task. Earch room had its own "placement" of windows; windows could appear in more than one room with position, size and iconification (regular, minimize, maximize) possibly different in different rooms.

Shaping

Users could create rooms (desktops), add windows, and set the room up for an intended task. By so doing, users shaped their working environment so that future work would be easier to carry out. This is much like choose preferences, but it is done over an open-ended set of applications that can be deployed on documents in windows. Shaping may be thought of as a special case of constructing with a construction set in which the glue (the language describing constructions) is not directly expressed. Rather, configurations are expressed extensionally—by placing things where you want them to be. This is much like using blocks, erector sets, Lego or electric trains.

Indirect shaping

Although configurations could be designed and set up, the usual practice with Rooms was to just do one's work in the room appropriate for it, adding, moving and removing documents as the need arose. Rooms simply remembered where things were (and what size, and what iconization) before leaving a room, and re-established them in those positions when the player returned. So the player did not ever have to explicitly express or even focus on the configuration of rooms and windows. The system was shaped as a direct but tacit consequence of their actions. As

a pure side-effect of thier work, the tailoring in effect cost the player nothing. But unlike "smart" or adaptive systems, the complete state of the system was directly and unavoidably accessible (visible) whenever it was needed (the player was in the room).

BUTTONS

In Rooms, one way of requesting transitions to other rooms was by clicking on doors. But doors were implemented as a special use of a more general notion invented for Rooms, called Buttons [7,8]. A button was a window with presentation that made it look like a button, and action expressed by any Lisp expression which would be evaluated when the button was clicked. Doors had a door-ish presentation, and an action of transitioning to another room.

Work on Buttons was continued at EuroPARC, where means of sending buttons in documents, including e-mail, enabled a culture of sharing actions captured in buttons to develop. This development took place over a number of years as a result of "extraction", an interative style of brain-surgical tailoring.

Extraction

As buttons were used, patterns of practice became things that I, as the original designer and continuing developer of Buttons, wanted to recognize and make easier for users to take into their own hands—and tailor to their needs. For example:

1. Players wanted to create buttons with varying presentations to both reflect different semantics (e.g. doors, mailbox activity), and to distinguish them from one another. Initially, all that was available was designating a fixed image. As usage made clear the ways players wanted to think about this, a compositional language (construction set) for describing an open-ended set of button-like presentations was defined and added to the Buttons package. The prior expression (an image) was a legitimate expression in this language, thereby insuring all older buttons continued to behave as expected.
2. Players wanted to make modify the actions associated with buttons. As the actions were expressed as Lisp exrpessions, this was apparently beyond the reach of those who did not have command of Lisp. Or so we thought, until we saw users editing these expressions (which they did not understand) by picking out key pieces (like filenames, or dates, or names) that they did understand, and replacing them. This suggested that we should honor this practice by enabling action creators to identify subexpressions likely to be changed (identifying parameters) and providing ways of making such changes without having to deal with the expression itself. Doing this extracted this behavior and made it more directly available to the players.
3. Specifying values for parameters for actions initially was presented to players as the ability to provide anything they wanted. It was quickly discovered that every knew the type that they value should be, and the users wanted editors for these values that took advantage of these types. Mechanisms for defining types, specifying editors for those types, and selecting among applicable editors on the basis of the context were added to Buttons. This required action creators to express a little more about their understanding of the action, thereby providing players with much more semantically meaningful ways of tailoring their buttons to suit their circumstances.
4. Players began to change the presentation of buttons to reflect the progress of the actions they provided (giving good feedback to the players). This pattern was noted, extracted and honored by making the expression of the presentation a parameter that was responded to immediately it was changed.
5. Further programming-like mechanisms were added to Buttons to capture at a non-Lisp level some of the common ways of factoring action descriptions. Subexpression-valued parameters were introduced to permit modularization of action expressions. A way of computing values of parameters at time of use were introduced, extending the context-awareness of button actions. And the special case of interacting with the player to get a value when it was needed extracted a common pattern of building interactive buttons.
6. Buttons became a common way for players to make requests for doing most oft-repeated actions. As might be expected, the pattern of repeating an action at regular intervals soon emerged (e.g., every morning read your mail, or update a calendar). Upon reflection, Buttons were extended with the addition of an interval for repetition, the default case being never (no repetition—embedding the old behavior as the default case). not only did this support the observed behavior, it enabled the immediate emergence of rich set of clocks (action repeated every second, or every minute; action to replace the presentation with some expression of the current time, somewhere).

And so the process of extracting from behavior the patterns and honoring them with backward-compatible brain surgery permitted the emergence and honoring of player-driven patterns.

POSITION

Tailorability means providing for the players the capacity to impress upon the system the behavior they want of it. Seven different forms of, and mechanisms for tailoring have been broken out of the experience with three research systems. These mechanisms have proved somewhat successful in achieving some measure of tailorability.

However, in the longer run, I think we need much more. We still have no means to get a system to change its mind about how it views the world to a view unanticipated by the designer, short of brain-surgery, which generally is available only to those with access to, and familiarity with, the implementation.

What is needed are systems that begin to deal much more explicitly in their own registration of the world, the presentation of that registration to the players, and the

alteration of that registration to reflect the players changing world. Further, since players will differ in their understandings of the world, these systems will have to engage in simultaneously holding the multiple different truths that different players espouse, and actively supporting the negotiation of those different truths when they encounter one another.

REFERENCES

1. Henderson, A., Kyng, M., There's No place Like Home; Continuing Design in use; in Greenbaum, J., Kyng, M.: Design at Work - Cooperative Design of Computer Artifacts, LEA, Hillsdale, pp 219-240. [1991]
2. Henderson, D.A. Jr.,. Trillium: A Design Environment for Copier Interfaces, (videotape presentation), CHI'83 (Boston MA, May 1983).
3. Henderson, D.A. Jr.,. The Trillium User Interface Design Environment, in Proceedings of CHI'86 (Boston MA, May 1986), ACM Press, 221-227.
4. Blomberg, J.L. and Henderson, A., Reflections on Participatory Design: Lessons from the Trillium Experience, in Proceedings of CHI'90 (Seattle WA, April 1990), ACM Press, 353-359.
5. Henderson, D. A. Jr., Card, S.K., Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-based Graphical User Interface, ACM Transactions on Graphics, Vol. 5, No. 3, July 1986, pp 211-243 (Published July 1987).
6. Card, S.K., Henderson, D. A. Jr., A Multiple, Virtual-Workspace Interface to Support User Task Switching, in Proceedings of CHI'87, Human Factors in Computing Systems, Toronto, Canada (1987).
7. Robertson, G. G., Henderson, D. A. Jr., Card, S. K., Buttons as First Class Objects on an X Desktop, in Proceedings of UIST'91, Symposium on User Interface Software and Technology, Hilton Head, SC, USA. (1991)
8. Maclean, A., Carter, K., Lovestrand, L., Moran, T., user-tailorable Systems: Pressing the Issue with Buttons, in Proceedings of CHI'90, Seattle, Washington [1990].