Avoiding Unintentional Inconsistency

Gary Perlman
OCLC Online Computer Library Center
6565 Frantz Road; Dublin, Ohio, USA 43017
Perlman@oclc.org

INTRODUCTION

Consistency sounds like something to strive for. After all, who would strive for inconsistency? Consistency is widely praised in guidelines and principles, but with some scrutiny, it becomes clear that consistency is hard to define and measure. Kellogg discusses several dimensions of consistency (including platform/devices) [kell87, kell89] and both Kellogg and Grudin [grud89] note that a design can be internally consistent within an application, or externally consistent with other applications. Grudin sums it up well by noting: "Thus, there may be no simple approach to determining the relative significance of consistency along various dimensions and levels." (p.1172) Caulton and Dye [caul97] concluded that consistency between applications is less important that task-appropriateness when applications are specialized.

Although consistency may be hard to design for and measure, problems of consistency may be easier. Reisner [reis87, reis90] attempts to formally describe inconsistency to predict where users will have problems using a system. Grudin [grud89] gives several informal examples where inconsistent design choices are more usable than consistent ones, and any designer can tell of cases where rules have been broken to address a specific user-task need. Given that, I will define "unintentional inconsistency" as the situation where two parts of a design differ for no good reason. For example: different terms used for the same concept; different layouts on different displays; different functions available in equivalent contexts. All these are unintentional, mind you, and probably due to limitations in resources, tools, techniques, and so on.

PROJECTS

I have been involved in several projects in which avoiding unintentional inconsistency was a primary motivation: SETOPT [perl85a] generated manual entries and parsers for UNIX command line options, with the goal of using the same information for both. The techniques used in SETOPT were formalized in [perl85b] and [perl89]: templates for targets (devices, displays) were instantiated with values of several variables. A change of a template resulted in global (consistent) changes; a change in variable would result in changes wherever the variable was used. Even without templates, using rules for display, objects could be rendered automatically to create displays [perl87].

More recently, the techniques have been applied to the design of FirstSearch, an online research service [perl00, perl02]. The system architecture - created with the primary goal of being able to change the design with inevitable changes in requirements - separates semi-structured [perl93] information into functional, display, and language partitions. Structured information is inserted into templates that dynamically generate displays in multiple languages, for multiple platforms, for multiple user types with individual preferences.

The technique is flexible. New translations of the service have been added by translating about 9000 partitioned words and phrases into Arabic, Chinese (2 dialects), French, Japanese, Korean, and Spanish, generally with no changes to the display or functionality. Even within English, the partitioning helps promote consistency in language usage. By adding new templates for larger displays, new platforms have been accommodated in a few hours of work including Web TV and the character-based Lynx browser. By adapting the main template and its components, most Section 508 accessibility requirements were met, leaving remaining requirements to changes to functional areas. Minor limitations of some devices have been accommodated with minimal specifications, for example: early Netscape browsers could not display Greek entities like α, so they could be shown as text (alpha) instead of a symbol (α):

```
[browser=Netscape4]
alpha = alpha
beta = beta
...
omega = omega
```

More substantial display limitations are accommodated with conditional display of functional components; all attributes of objects are available during display generation.

MEASUREMENT

Along with the partitioning of structured information comes the ability for metrics on that information. Metrics have been used to avoid unintended inconsistency. A simple example is to measure, for each term (e.g., Search), how many times it appears in values compared to how many times a reference to it appears. A more sophisticated example involves checking device-specific templates to ensure that they contain the same references to parts to

include; this has been used during "facelifts" to the user interface.

CONCLUSIONS

Some software development techniques are especially well suited to ensuring consistency in user interfaces, even while allowing flexibility in the efficient specification of inconsistency where the designer intends it. I do not think that cross-platform development presents any different challenges than, say, customizing for different types of users. What is not clear to me are the *types* of changes that are necessitated by cross-platform development compared to other dimensions of change. Being able to anticipate those would help estimate and allocate resources.

REFERENCES

The format used to the references in this essay are intentionally inconsistent with ACM style.

[caul97] Caulton, David A. and Dye, Ken. Do Users Always Benefit When User Interfaces Are Consistent? Proceedings of the HCI'97 Conference on People and Computers XII, 57-66, 1997.

[grud89] Grudin, Jonathan. The Case Against User Interface Consistency. Communications of the ACM, 32:10, 1164-1173, 1989.

[kell87] Kellogg, Wendy A. Conceptual Consistency in the User Interface: Effects on User Performance. Proceedings of IFIP INTERACT'87: Human-Computer Interaction, 389-394, 1987.

[kell89] Kellogg, Wendy A. The Dimensions of Consistency, in [niel89].

[niel89] Nielsen, Jakob (Editor). Coordinating User Interfaces for Consistency. New York: Academic Press, 1989.

[perl85s] Perlman, Gary. SETOPT: A UNIX Command Line Options Parser Generator. Proceedings of the Winter USENIX Conference, p.160-164, 1985.

[perl85b] Perlman, Gary. Multilingual Programming: Coordinating Programs, User Interfaces, On-Line Help and Documentation. ACM Fourth International Conference on Systems Documentation, 123-129, 1985.

[perl87] Perlman, Gary. An Axiomatic Model of Information Presentation. Proceedings of the Human Factors Society 31st Annual Meeting, 1229-1233, 1987.

[perl89] Perlman, Gary. Coordinating Consistency of User Interfaces, Code, Online Help, and Documentation with Multilingual/Multitarget Software Specification, in [niel89].

[perl93] Perlman, Gary. Information Retrieval Techniques for Hypertext in the Semi-Structured Toolkit. Proceedings of ACM Hypertext'93, 260-267, 1993.

[perl00] Perlman, Gary. The FirstSearch User Interface Architecture: Universal Access for any User, in many Languages, on any Platform. Proceedings of the 2000 ACM Conference on Universal Usa bility, 1-8, 2000.

[perl02] Perlman, Gary. Achieving Universal Usability by Designing for Change. IEEE Internet Computing, 6:2, 46-55, 2002.

[reis90] Reisner, Phyllis. What is Inconsistency? Proceedings of IFIP INTERACT'90: Human-Computer Interaction, 175-181, 1990.

[reis93] Reisner, Phyllis. APT: A Description of User Interface Inconsistency International Journal of Man-Machine Studies, 39:2, 215-236, 1993.