Task-Interface Matching: How We May Design User Interfaces

Wan Chul Yoon
Department of Industrial Engineering, KAIST
wcyoon@mail.kaist.ac.kr

This paper proposes a comprehensive framework to view and manage the design process in a cognitive engineering perspective. UI designers, rather than following a rigorous top-down task-based design process, often adopt more casual approaches that utilize their prior knowledge and existing solutions. Besides the obvious efficiency and practical reliability, the bidirectional and opportunistic strategies have their own serious grounds. This paper starts with a detailed analysis of those practical strategies to reveal the principles of constraint-based approach. Based on the findings, task-interface matching, a comprehensive relational framework for UI design is suggested and discussed. The designer can examine such concepts as organization, semantic signification, consistency, task affinity, schemata, and metaphor within the framework. Based on task-interface matching framework, a systematic and comprehensive design process that aims for efficient and more knowledge-rich usability design is proposed.

INTRODUCTION

As the electronic devices, software programs, and web sites grow to embrace greater functionality and complexity, the benefit of systematic interface design processes such as task-based design receives more emphasis. A typical task-based design approach would start the procedure with eliciting the user's task needs and task knowledge, goes through a conceptual interaction design, and ends with a concrete, physical design of the interface. In practice, however, designers seldom follow the top-down sequential process rigorously. A more common practice is the heavy reuse of the known partial design solutions for the sake of efficiency and, in many cases, reliability. In entertaining, evaluating, selecting, adapting, and composing the known partial solutions, designers tend to manifest bidirectional and opportunistic search behavior (Guindon, 1992; Visser, 1996; Preece et al., 1994, Yoon, 2001). The strategies may be regarded random and casual but in fact highly rational considering the design environment.

Being Bidirectional

Being bidirectional, the search for design solutions may in part proceed in a bottom-up direction, first starting with a set of very likely interface elements and then advancing to good organization of them and interaction methods that use them. Some may be concerned that the early entertaining of interface solutions could lead to a premature design due to cognitive anchoring. However, there are also several good reasons for UI designers to partly follow the bottom-up direction.

First, starting with concrete examples is cognitively easy and efficient. Researchers observed that the designers, even while considering design guidelines, tended to depend heavily on pictorial examples rather than detailed explanations in the guidelines (Thovtrup and Nielsen, 1991). The efficiency advantage of such example-based or case-based design approach weighs more when time and resources for product development are restricted, as it is the case nowadays more often than not. It may even produce better and more reliably working design solutions when the time pressure is high.

Second, the UI designer must put oneself in the place of the user to design a usable interface (Meister and Farr, 1967; Hix and Hartson, 1993) and the user’s strategy of learning the interface is often bottom-up. Klein (1989) suggested RPD (Recognition-primed decision making) as a general decision-making strategy, in which the user recognizes the pattern of given cues and, through prior experience, categorize the situation. A great body of research also supports the view that users of interactive systems rely on 'exploratory learning' (Rieman, 1996). Since the user’s exploration is prompted by the visible and operational interface, selecting concrete interface means may have to precede determining more abstract interaction methods.

Third, perhaps the most inevitable reason, which is also related to the above, is that the designer must comply with the socially settled *de facto* standards of various interface objects and methods to insure proper communication with users. It means that there exist little room for designer’s free choices at the level of physical interface. The design freedom is also severely limited at the highest, task level because the tasks are usually defined by the missions of the system. Since design difficulty is generally reduced where there are more constraints or fewer choices, the planning search is more efficient when starting from the top and the bottom levels of design.

Being Opportunistic or Constraining

Ball and Omerod (1995) critically examined various possible meanings of opportunistic design process. They argued that the term *opportunistic* processing in the genuine sense should be used to refer to leaping to a design problem that are not related, in a goal-means path, with the problem that is being currently dealt with. Even in this rigorous sense, the last two reasons for the bottom-up processing make UI design particularly more opportunistic than other software or product design areas.

In designing UIs, the design freedom is severely constrained at both top and bottom abstraction levels. The user tasks to be designed are usually given as the system purpose and largely determined by the prior task knowledge that the users are assumed to have. On the other hand, at the most concrete interface level, the designer has to comply with the user expectation regarding the shapes and working of interface means according to well-accepted social standards (Yoon 2001). According the strategy of *constraint propagation* in artificial intelligence, the search is much more efficient when the decisions are made starting from the most constrained problems and proceeding to the freer ones, thanks to the minimization of random guesses. In UI design practice, designers entertain almost certain solutions into the partial solution set as soon as they are found during the process and exploit them in turn to narrow down solutions for the adjacent problems (Figure 1). Since the certain solutions are scattered, the designer is observed to leap around problems collecting solutions in an opportunistic manner. In short, opportunistic processing in many cases may not be a mere deviation from the rational design process but itself a sensible strategy that delivers believable design efficiently without much loss of output quality. Not to mislead, such strategy should be referred to as constraint-based process rather than opportunistic process. In this mode, 'premature' search occurs only when mature solutions are found.

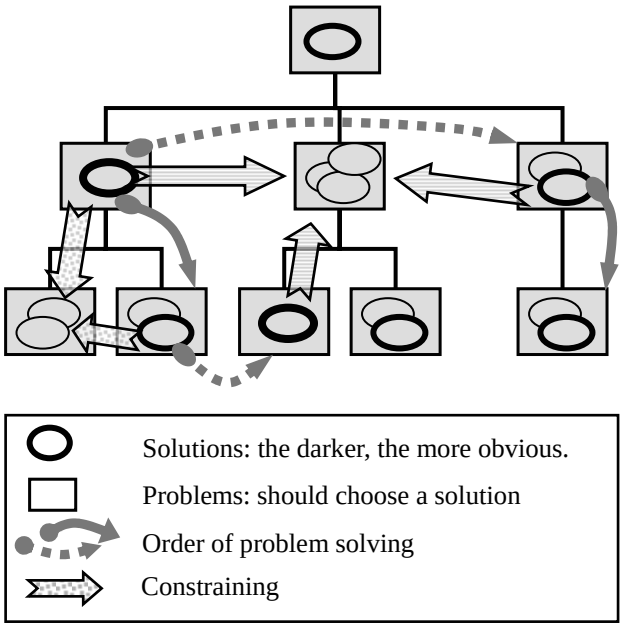


Figure 1. The design flow and constraint propagation

Figure 1 shows a goal-means tree of design problems. In a top-down design, an alternative that is chosen for a parent problem affects the decision-making in its child problems. However, for some problems, the solutions are quite obvious, as denoted by the thicker ovals. The designer may follow the arrows to entertain the obvious solutions as early as possible. The dotted arrows denote the leaps to other problems of which obvious solutions somehow draw the designer's attention. While those may be called opportunistic, the strategy is far from randomness since thus augmented set of obvious solutions is effectively used to constrain the decision-making in adjacent design problems.

It should be noted that, as discussed earlier, the obvious solutions are more densely distributed at the top, task level and the bottom, interface level. It induces in turn the tendency toward bidirectional search from both ends.

TASK-INTERFACE MATCHING

A user interface provides 'knowledge in the world' to the user (Norman, 1988) that can be used as cues or reminders for more abstract knowledge such as possible tasks or appropriate sequence of actions for a task. In the opposite direction, task knowledge is used to predict the availability or behavior of interface elements. Such evidence-hypothesis relationship in a cycle characterizes the use of software and electronics interfaces (Figure 2)

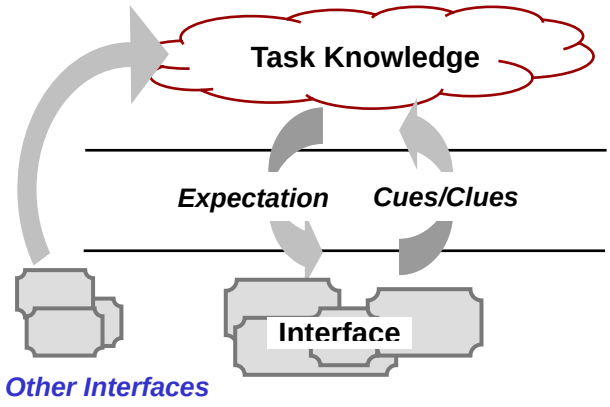


Figure 2. Task Knowledge and Interface

The designer's bidirectional and constraint-based problem solving can actually be integrated in the framework of this *task-interface matching* cycle. The matching is not simple coupling of tasks and interface means but is concerned with a very comprehensive relational structure of user interfaces. Within the structure included are design constructs with organization, relations among them across the abstraction levels, and the constraints on arranging the relations. The

top-down expectation and bottom-up cues are reflecting the matching relations.

The design constructs at different abstraction levels are shown in Table 1. They may have frameworks or organizations that put them in contexts. The organizations are frequently used for relating the lower level constructs to the more abstract ones.

The following typical matching relations across the levels are identified as follows.

Groups: Task groups by hierarchical structure or affinity should match visually grouped interface objects and vice versa.

Syntax: Tasks with high affinity should share consistent methods/sequences (task-to-interaction);
A prominent interface object should have consistent autonomous syntax around its use.

Semantics: Interface objects/operation should have consistent and congruent signification;
Labels should rightly signify the corresponding functions.

Flow: Flow structure expected by task knowledge (i.e., order, branching, looping) should be reflected in interaction flow and the availability, visual layout, prompts, and affordance of the physical objects.

Table 1. Design Constructs and Organization

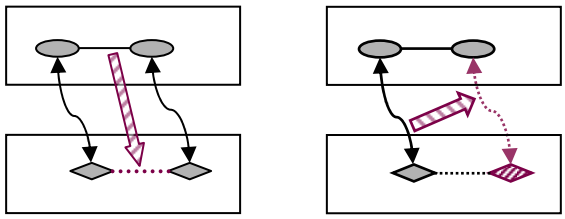
Abs.Level	*Constructs*	*Organization*
Task	Tasks, Subtasks	Goal-means tree, Task affinity. Task flow
Interaction	Syntax	Flow structure, Consistency
	Semantic operation	Mental model, Task-function relations
	Decision making	DM context, Awareness, Feedback
Physical Interface	Control, Display	Grouping, Layout, Affordance

Design solutions can be described in term of the chosen constructs and/or their established organizations. The set of hypothetical design solutions that are so far entertained constrain the adjacent design decisions through a few constraint rules reducing the complexity of remaining problems. The designer can therefore follow these constraints to reduce the complexity of the design problem solving as soon as possible avoiding premature choices. The following types of constraints are quite general:

Vertical parallelism: If a relation between design solutions across abstraction levels is adopted, then the same relation should also be applied between the design constructs that are closely related to the solutions. (e.g., if the left arrow button is assigned to ‘previous page’ operation due to the icon’s likely signification, the right arrow button should also be assigned to ‘next page’ operation.) (Figure 3 Left)

Horizontal parallelism: If an organization is established among design constructs at an abstraction level, and there exist a common relation definable on the constructs and projected to another abstraction level, then there should also be some corresponding organization among the constructs at the other level. (e.g., if two functions are perceived as a pair, the buttons to activate them should be located side by side and, if applicable, in that order.) (Figure 3 Right)

Exclusion: A design solution, if accepted, may eliminate some possible solutions of other design problems. (i.e., having assigned a button to page navigation, one cannot use the same button for line navigation in the same mode.) Exclusion may work in combination with other constraints.



paralleling horizontal relations

paralleling vertical relations

Figure 3. Parallelism Constraints

The user deals with an interface assuming that the designer designed the interface with such constraints to maximize task-interface congruence and minimize the possibility of confusion. Expecting this user belief in turn, the designer should fulfill it, and hence, the constraints come into effect.

COMMUNICATION, METAPHOR, AND SCHEMA

As we have already used terms from linguistics in this paper such as signification of interface objects (i.e., signs) and syntax, a user interface can indeed be regarded as a *communication* in the proper meaning (Nake and Grabowski, 2001). Like a language, the designer displays interface objects (*cf.* words) in a context that is expressed by locating, grouping, ordering, and other means to deliver the designed system image. From the designed interface, the user should infer what the artifacts are, how they are used, and how they are integrated with the context (Salles et al., 2001)

Designers and users need to assume a common understanding of the signs, their signification, and contexts to communicate via an interface. The interpretation of interface constructs and their organization may, however, deviate from what were meant by the designer due to differences in schemata or mental models. Metaphors in user interfaces may be viewed as an instrument to secure a

common schema between the designer and the user. Once established, such common schema can faithfully deliver a batch of relations and organizations that have been adopted in the design to the user. The task-interface matching framework can provide a ground for devising appropriate metaphors or considering the effects of popular schemata.

DESIGN PROCESS FOR TASK-INTERFACE MATCHING

Based on the framework of task-interface matching, a natural design process emerges. The process is very practical that UI designers in many domains can be said to be already using its simplified version albeit in an implicit way. Its power is in the explicit handling of relations and abstract constructs to support the naturalistic decision making and knowledge management.

1. Analyze the tasks. Identify the task tree and assess task affinities.
2. Collect the inventory of likely interface objects and means. The inventory should be larger than the set of currently used ones. Among them, identify obvious choices. Study their likely signification considering the target users and the culture.
3. Collect widely expected interaction methods to achieve the tasks using the objects. Assess the degrees of acceptance and expectation by users.
4. Identify possible metaphors, schemata, or mental models as necessary.
5. Solve the design problems. Start with the set of the most obvious solutions and follow the constraining paths. For every problem, record the solution together with the applied task-interface matching relations, organizations, schemata or metaphors, and adjacent solutions that participated in constraints.

CONCLUSIONS

The task-interface matching framework provides a unified perspective on the problem solving of the designer's and the user's, combining both in a communication cycle. The framework grasps and economically describes the essential relational structure.

Within the framework, the design problems and solutions are stated in terms of the relationships, including vertical abstraction and horizontal organization, which is the most important characteristic of the approach. Many usability concepts such as consistency, congruence, grouping, and metaphors can only be described in the network of those relationships. The relationships have been recognized and dealt with in usability evaluation but not used in a constructive manner in design. The explicit handling of those relationships will allow designers systematically pursue solutions for usability during the designing rather than put the issues off for later evaluation.

A naturalistic but systematic design process is proposed based on the framework. The design process implements bidirectional search and constraint-propagation strategies for efficient construction of coherent interface designs. Principles and practice may have a wider contact in this process.

ACKNOWLEDGMENTS

This research was supported by KOSEF (Korea Science and Engineering Foundation) and ARIEL, France.

REFERENCES

Ball, L.J. and Ormerod, T.C., 1995. Structured and opportunistic processing in design: a critical discussion, International Journal of Human-Computer Studies 43, 131-151.

Guindon, R. 1992. Requirements and design of DesignVision, an object-oriented graphical interface to an intelligent software design assistant, In *Proceedings of CHI'92*, 499-506. New York: ACM Press.

Hix, D., Hartson, H.R., 1993. Developing user interfaces: Ensuring usability through product and process. Wiley, New York.

Klein, G. 1989. The recognition-primed decision making. Advances in Man-Machine Systems Research, 5, 47-92.

Meister, D., Farr, D.E., 1967. The utilization of human factors information by designers. Human Factors, 9, 71-87.

Nake, F. and Grabowski, S., 2001. Human-computer interaction viewed as pseudo-communication, Knowledge-based Systems, 14, 441-447.

Norman, D.A., 1988. The psychology of everyday things, Harper & Row, New York.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T., 1994. Human-computer interaction. Reading, MA: Addison-Wesley.

Rieman. J., A Field Study of Exploratory Learning Strategies, 1996. ACM Transactions on Computer-Human Interaction, 3(3), 189-218.

Salles, J., Baranauskas, M.C., and Bogonha, R.S., 2001. Towards a communication model applied to the interface design process. Knowledge-Based Systems 14, 455-459.

Thovtrup, H., Nielsen, J., 1991. Assessing the usability of a user interface standard. Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems, New Orleans, LA, 335-341.

Visser, W., 1996. Use of episodic knowledge and information in design problem solving. In N. Cross, H. Christiaans, and K. Dorst (Ed.), Analysing Design Activity, .271-289. New York: Wiley.

Yoon, W.C., 2001. Identifying, organizing and exploring problem space for interaction design. 8th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems, Kassel, Germany, 81-86.