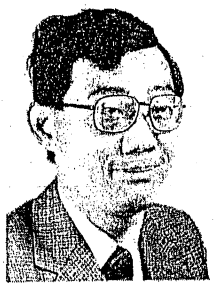
Techniques

User Interface Design for Decision Support Systems: A Self-Adaptive Approach

Ting-peng Liang
Department of Accountancy, University of Illinois at Urbana-Champaign, 1206 South Sixth Street, Champaign, IL 61820, USA

This paper presents a self-adaptive approach to user interface design. The primary philosophy of this design is that the user interface must be aware of the changes in its user's behavior and then adapt to it. Three different default policies are proposed to control the adaptation of a user interface: fixed default, dynamic default, and no default. Performance of these policies are compared for various patterns of usage. Mechanisms that determine the optimal default value to reduce the unnecessary effort are also discussed.

Keywords: User interface design, Decision Support Systems, Self-adaptive design, Intelligent systems, Intelligent user interface.



Dr. Liang is an Assistant Professor at the University of Illinois at Urbana-Champaign. He holds Ph.D. and MA degrees from The Wharton School of the University of Pennsylvania, an MBA from National Sun Yat-sen University, and a BS in Engineering from National Cheng-kung University (both in Taiwan, ROC). He has published many articles in research journals, including *Database, Decision Sciences, Decision Support Systems,* and *Journal of Management Information Systems.* His current research interests include design and implementation of decision support systems, model management, and expert systems applications.

North-Holland
Information & Management 12 (1987) 181–193

1. Introduction

A computer-based decision support system (DSS) is designed to improve unstructured or semi-structured decision making. It has three major components: an interactive user interface, a database management system, and a model management system. Because the user interface is the channel through which a user communicates with the DSS, much of the power, flexibility, and usability of the system depends on this interface.

Research in user interface design has increased dramatically in the past decade. Most of it focused on one or more of the following topics:

1. *Design Processes*: e.g. the ROMC approach [28], verb-oriented design [16], cognitive considerations [26], and the nature of man–computer interaction [9,22].
2. *Representation formats*: e.g. graphs or tables, audio or video, color or not [7].
3. *Dialog styles*: e.g. Q/A style, menu selection, input/output form, or other fancy input instruments such as touch screen and mouse [2,20,22,27].
4. *Interface functions*: e.g. help, tutorial, and error messages [2,20].

Some preliminary findings and considerations have been presented [14,19,29].

Recently much interest has been developed in building adaptive interfaces [6,8,25,30]. In general, adaptation refers to the ability of the system to act appropriately in a given context. For example, an adaptive system may make available tools relevant to a particular task and change their functionality to suit individual preference.

Traditionally the system designer has been responsible for designing the user interface based on pre-determined requirements. However, because continuous evolution is a major feature of DSS [12,16], a static interface designed in the tradi-

tional approach can hardly provide the required flexibility. Therefore, interfaces that allow easy adaptation to different user requirements are highly desired. An empirical study also indicated that adaptive design led to higher user satisfaction [1].

There are three approaches to designing an adaptive interface:

1. user-involved adaptive design,
2. user-controlled adaptive design, and
3. self-adaptive design.

The user-involved adaptive design requires that the user be involved in the system adaptation process [10]. The designer and the user cooperatively specify and develop the first version of the system; then the user evaluates the system with the help of the designer. If the system is valuable, the user accepts it. Otherwise, the designer examines the user's behavior and modifies the system accordingly.

Instead of having the designer develop the system with the assistance of the user, the user-controlled adaptive design allows the user to specify the desired interface with an interface generation language [4]. The system then creates the interface based on the specifications. In this approach, the user is fully responsible for the adaptation of the system.

Although both the user-involved and the user-controlled approaches provide the system with flexibility, they place too much burden on the user. The self-adaptive design allows the system to adapt to the anticipated change automatically and enables the system to share the responsibility for adaptation. Its basic premise is that some changes in a user's behavior are predictable and, therefore, the system should handle them automatically.

This paper explores issues in developing a self-adaptive interface and presents an approach that controls automatic adaptation by keeping track of user profiles and adjusting system defaults. A self-adaptive interface developed with this approach has two major features: it presents personalized interfaces to different users and it updates its interface according to change in the user's behavior.

2. Overview of General Guidelines

Since many issues are unsolved, design of an effective interface remains an art rather than a science [23]. However, some general guidelines do exist. They provide a basis for good interface design. In general, these guidelines fall into three categories: computer technology, task requirement, and user characteristics.

Recent advances in computer software and hardware, such as multiple windows, mouse, touch screen, and audio input and output, have significantly increased the diversity and flexibility of human–computer communication. Therefore, the application of appropriate technology is essential to effective interface design. Here "appropriate" implies that the designer must consider other relevant issues in determining what technology best serves the user. The most up-to-date technology is not necessarily the most appropriate one. For example, audio inputs and outputs are good in some situations but may not be appropriate in the situation where confidentiality is important or a quiet working environment is required.

Different tasks and users may also have different requirements. For example, empirical studies have indicated that tabular output outperforms graphics in simple jobs or situations where accurate numbers are important; but graphic presentation is better if the job needs comparison, trend analysis, or complex information [7]. In addition, cognitive styles may also influence a user's preference for presentation formats [3].

Taking these three classes of factors into account, a good interface must fulfill the following requirements:

1. Be diverse: support both inexperienced and experienced users.
2. Be forgiving: have good error recovering capabilities.
3. Be efficient: minimize the effort required to accomplish a job.
4. Be convenient: provide good accessibility to all operations.
5. Be flexible: provide multiple routes for accessing an operation.
6. Be consistent: minimize learning requirements and unexpected actions.
7. Be helpful: provide good help and error messages.

A summary of the guidelines is given in Table 1. Since some recommendations are conflicting (e.g. consistency and flexibility), effective interface design involves delicate tradeoffs [19].

Table 1

1. *General Principles*
   - Provide sufficient functionality to make the system useful
   - Have appropriate user involvement in the development process
   - Allow user control of the system
   - Provide support to different levels of users, including experienced and inexperienced users
   - Provide appropriate combination of input/output devices.
2. *Input Design*
   - Provide multiple input channels, e.g. voice recognition, mouse, etc.
   - Use full-screen-oriented interface for data entry and retrieval
   - Check input integrity, including confirmation of exceptional inputs and error checking
   - Minimize the effort required to complete an operation.
3. *Output Design*
   - Provide multiple information channels, e.g. screen, printer, etc.
   - Represent information in appropriate format to enhance decision performance
   - Use non-verbal signals, such as reverse video, beep, and flashing characters
   - Respond in a reasonable time
   - Provide powerful report generator and use windows.
4. *Interface Functions*
   - Provide powerful commands, which include
     - support command synonyms and abbreviations
     - provide productive syntax
   - Aid the transition from novice to expert
   - Allow the user to "macro-ize" or customize tasks
   - Inform the user of the functions available in the system.
5. *User Control*
   - Allow reversible actions when mistakes are made
   - Provide immediate feedback when a mistake is made
   - Allow the user to stop processing at any response point without jeopardizing system integrity
   - Provide information on current state, what is required next, and what had happened previously.
6. *Interaction Style*
   - Provide multiple interaction styles, e.g. menu selection, commands, query-by-example, etc.
   - Use consistent naming conventions and formats
   - Provide near-natural language communications.
7. *Help and Error Messages*
   - Provide on-line tutorial aids and help facilities
   - Provide sensible defaults to reduce the need for learning
   - Provide sensible error messages when mistakes are made
   - Provide short-cuts across menu-levels and in question/answer mode
   - Provide command syntax prompt and functions that allow the user to edit and save command strings.

3. Adaptive Requirements of DSS

In addition to these, system adaptability is important to user interface design for DSS. It is particularly necessary when the user needs and expectations are likely to change frequently. On one hand, adjusting an adaptable system is much cheaper than designing a new system; on the other, an adaptive interface has advantages [21], such as:

1. It can be tailored for individual differences,
2. It enables users to carry out their tasks more effectively, and
3. It allows the system to support users with different levels of experience.

Sprague and Carlson [28] divided system flexibility into four levels: changeable, adaptable, modifiable, and evolutionary. From another point of view, since a DSS integrates information and computer technology to support human decision making, it needs three kinds of adaptation: to a new problem or task, to a new technology, and to different user behavior.

Based on current technology, it is almost impossible to build a system which can self-adapt to a change in problem domain or new technology. For example, a system is unlikely to install a more powerful graphics terminal automatically; equally, a DSS cannot solve a new problem satisfactorily without being revised by the user or designer. However, by implementing parameter adjustment mechanisms which change system behavior by adjusting some parameters [24], it is possible for the system to adapt to a changed behavior without requiring the involvement of either the designer or the user (e.g. [9,13,17]).

For example, Croft [6] proposed a self-adaptive document retrieval system to improve the effectiveness of document retrieval. It selects appropriate search strategies based on user feedback and the context of a query. Mason [21] also reported a command prompting system designed for enhancing the learnability of an interactive system. This self-adaptive capability is particularly important in developing a DSS generator. It allows the DSS generator to tailor specific DSSs for multiple users.

4. Design of a Self-Adaptive Interface

To implement the parameter adjustment approach in a user interface design, the following three issues are essential [18]:

1. *How to identify and classify users;*
   Users must be categorized in order to determine their requirements.
2. *How to determine appropriate actions;*
   Knowledge about appropriate actions for a particular type of user is required.
3. *How to control the adaptation process;*
   Mechanisms for controlling the system adaptation process are required to ensure evolution in a desired way.

4.1. Patterns of Usage

Although design of such an interface is not a very new idea, the area remains virtually undeveloped. One of the major reasons is that we do not have a valid normative user model to indicate interface features appropriate for a particular type of users [5]. The complexity of human beings makes the development of such models almost impractical.

Instead of concentrating on a general user model, system adaptation can also be achieved by adopting heuristic methods. Several techniques have been developed, including pattern-matching and debugging [5]. Unfortunately, they were primarily designed for relatively structured environments, such as error handling and advice-giving. To meet the adaptive requirements of DSS, the method must focus on the evolutionary nature of user behavior and allow the system to adjust itself according to the anticipated user behavior. In addition, the method must be transparent to the user. That is, it must avoid interrupting the user's decision process in the course of data collection and system adaptation. Based on these criteria, inferring a preference by analyzing previous usage is natural.

The evolution of user behavior can be divided into three patterns: consistent, systematic, and random [18]. Users who prefer the same presentation of information are consistent users. Users who change their preference in a partially predictable way are systematic users. Users who change their preference unpredictably in a given situation are random users. Fig. 1 illustrates the difference among these patterns.

The reason that we classify patterns of usage is because this information can predict the anticipated behavior and then control the system adaptation by providing proper default policies. Although users may have different preferences, a user may also change usage behavior over time; e.g. a random user may evolve to a consistent user after several attempts. Therefore, patterns are not permanently associated with users. They are determined dynamically when the system is used.

```
1. Consistent user -- a user who prefers a particular
                      representation or operation
                      frequently.

        e.g., 1 1 1 2 1 1 1 1 1 1 ......

2. Systematic user -- a user who changes preference
                      systematically.

        e.g., 3 3 3 3 2 2 2 2 2 1 ......
        e.g.,

3. Random user -- a user who changes preference
                  randomly.

        e.g., 1 2 1 3 3 3 2 3 1 2 ......

Note:  1 ... bar chart
       2 ... line chart
       3 ... table
```

Fig. 1. Sample patterns of usage.

4.2. Default Policies

A default determines the action when no relevant information is provided by the user. A default policy is a strategy for assigning defaults. There are three such policies: fixed, dynamic, and no default (menu selection).

The *fixed default policy* offers a pre-determined output format (e.g. a scatter diagram) or dialog style (e.g. the prompt/response style) whenever an operation is activated. The *dynamic default policy* associates the default assignment with previous usage records. If the user asks for an operation or a representation different from the default, the new operation or representation will replace the old one. Depending upon a particular implementation, there are certainly other types of dynamic policies; e.g. we may define a two-change policy

that sets a particular operation as the new default after it has been requested for two consecutive sessions. The *no default policy* offers a fixed menu containing all available alternatives; this allows the user to select one.

Different default policies allow the system to behave differently. For instance, a system designed for a beginner may need the no default policy. After a period of learning, however, the system may detect a particular pattern of usage, say, the user always prefers pie chart presentation. In this case, the system should change to a fixed default policy with pie chart presentation as the default.

The advantage of the fixed default policy is that the user can correctly anticipate what will appear when an operation is initiated. The major disadvantage is that this may not reflect the user's present preference. The advantage and disadvantage of the dynamic default policy are exactly the opposite of the fixed default policy. That is, the system reflects the user's current preference, but it lacks consistency. The menu selection policy gives the user opportunities to select, but needs more actions and effort to obtain the result.

4.3. Mechanism for Default Control

In a self-adaptive interface, patterns of usage and default policies serve as parameters. We also need a mechanism to control the evolution of the interface. This mechanism collects usage records for each user, analyzes performance of various default policies, assigns the optimum default policy, and then integrates user interface elements (e.g. bar chart and pie chart subroutines) to build the interface. The user communicates with the system through the interface and at the same time provides information to the system for further evolution. *Fig. 2* briefly illustrates the role of this mechanism and the process of interaction.

Since a user's behavior changes over time, simply partitioning users into three patterns and then mechanically matching them with default policies is not adequate. We need a mechanism that dynamically offers default policies based on user actions. While developing such a mechanism, the following issues must be considered:

1. Design objectives and performance measures,
2. Rules for calculating performance, and
3. Rules for assigning default policies.

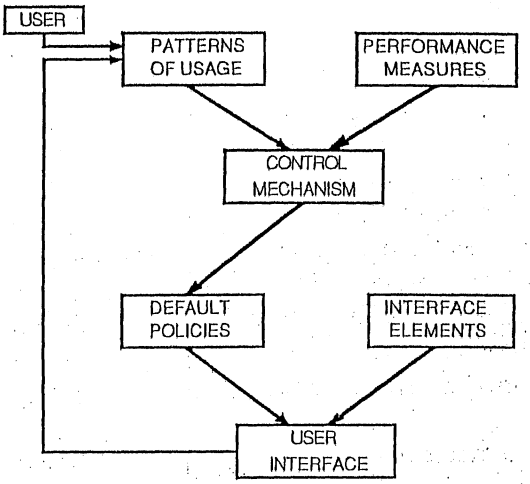


Fig. 2. Architecture of the self-adaptive mechanism.

4.3.1 Objectives and Performance Measures

The performance measures indicate how the design objectives have been achieved. Different objectives and performance measures may therefore result in different designs. In developing adaptive advice-giving systems, a common goal is to facilitate user learning in the interaction process (e.g. [5,6,12]). Since minimizing effort is one of the major considerations in developing effective interfaces, it can also be considered an appropriate goal for self-adaptive design. This goal is especially useful in information inquiry and presentation. Given this goal, the expected number of keystrokes required for performing an operation or presenting a piece of information may be adopted as a performance measure.

There are certainly other measures. For example, perceived consistency (the fixed default and menu selection policies are better than the dynamic default policy) and user control over the system (menu selection is better than the other two policies) are also good alternatives. A combination of multiple measures is also possible. In this case, however, a decision model will be needed to combine them.

4.3.2 Rules for Calculating Performance

Based on the selected measure, rules can be developed to calculate the performance of various default policies. Since these rules may vary from one implementation to another, we use an example here to demonstrate their functions. Suppose four users, *A*, *B*, *C*, and *D*, share a DSS that

presents an inventory report in four different formats. Because these users have quite different patterns of usage, the performance values and optimum defaults for various default policies are different, as illustrated in *Fig. 3*. The performance values in the Figure are expected numbers of keystrokes for obtaining the inventory report. They are calculated by dividing the expected number of keystrokes for a particular default policy by the total number of keystrokes of the no default policy (21, in this example). *Appendix 1* provides a PROLOG implementation.

4.3.3 Rules for Policy Selection

After determining the objective of design and measuring the performance of various default policies, rules are required to select the optimum policy. Development of these rules is also highly task-oriented. It varies substantially from one implementation to another. For analyzing the previous example, the following rules are used in the prototype of Appendix 1:

1. IF *the expected number of keystrokes of a policy is less than 0.5,*
   THEN *consider the policy as a candidate for default.*
2. IF *more than one candidate fulfills rule 1,*
   THEN *choose the one with the lowest expected number of keystrokes.*
3. IF *more than one candidate fulfills rules 1 and 2,*
   THEN *prefer fixed default policy to dynamic policy.*
4. IF *no candidate fulfills rule 1,*
   THEN *choose the no default policy.*

Based on these rules, the optimum settings for users *A*, *B*, *C*, and *D* can be determined. For users *A* and *D*, since no policy has performance less than 0.5, the optimum policy is no default. For user *B*, both the fixed default and the dynamic default policies have performance less than 0.5. According to rule 2 the system will accept the dynamic default policy. For user *C*, the optimum policy is fixed default; see *Fig. 4* for a sample session.

In summary, by providing the mechanism that examines usage records and controls default policies, an interface can be self-adaptive. This mecha-

Policies / Patterns	Fixed		Dynamic		Menu	
	Default value	Performance	Default value	Performance	Default value	Performance
A	2	0.57	3	0.67	no	1.0
B	2	0.43	2	0.23	no	1.0
C	2	0.29	1	0.57	no	1.0
D	3	0.71	3	0.91	n0	1.0

```
Note: Pattern A
      1 2 3 4 2 1 3 4 3 3 3 2 2 1 3 2 2 2 2 2 3

      Pattern B
      3 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 2 2 2

      Pattern C
      2 2 1 2 2 2 3 2 2 2 2 2 1 2 2 4 2 3 2 2 1

      Pattern D
      2 1 3 4 3 1 2 4 1 4 2 3 2 1 3 3 1 4 4 2 3

Where:  1 ... bar chart          2 ... line chart
        3 ... pie chart          4 ... table
```

Fig. 3. Performances of sample patterns.

```
- run(pattern_b).

- Default policy is fixed
  Default value is 3
  Performance is 0.905

- Default policy is fixed
  Default value is 1
  Performance is 0.667

- Default policy is fixed
  Default value is 2
  Performance is 0.429

- Default policy is dynamic
  Default value is 2
  Performance is 0.231


-------------------------------------------
    My Suggestions for pattern_b
-------------------------------------------

The optimum default policy is  dynamic
Current default value is  2
The expected performance of this setting is
0.231 keystrokes

-------------------------------------------
```

Fig. 4. A sample session.

nism has the following functions:

1. Keep track of the usage pattern of each user,
2. Compute the performance under various default policies,
3. compare the performance of each of the policies and select the best for the current pattern of usage, and
4. Assign the selected policy to the specific DSS automatically.

5. Applications of the Self-Adaptive Design

In the previous section, we described an application of the self-adaptive mechanism. In addition to this, there are other applications. For example, a DSS may tailor the sequence of information presentation for different users. Although we do not have enough evidence to conclude that the sequence of presentation may affect decision making, a good presentation sequence can increase the user's satisfaction and reduce unnecessary waste of time.

In determining the appropriate sequence of presentation, there are also three default policies: fixed, dynamic, and menu selection. A *fixed sequence policy* presents information in a fixed order, which will not be changed unless the default is changed. A *dynamic sequence policy* changes the order of presentation according to user behavior. In this case, the user's present action may affect the future presentation sequence of the system; e.g. if a dynamic policy is adopted and the user has chosen to look at net profit prior to viewing total sales, then next time the user invokes the system, net profit will be shown before displaying total sales. A menu selection policy offers a predesigned menu containing all available alternatives. No output will be presented until the user selects one.

Another interesting application of the self-adaptive mechanism is the development of a group DSS (GDSS), which supports a group of users. To meet various requirements of the users, adaptive and personalized interfaces are crucial.

In addition, the mechanism is also valuable in non-DSS areas such as word processing. Currently most wordprocessors adopt the fixed default policy which sets up a standard document format until the user changes it. Although this approach eliminates surprise, it could be tedious for some users if they have to change format every time they start to use the system. With appropriate application of the self-adaptive design, a wordprocessor could be more flexible; it may adopt a dynamic default policy that creates a new style sheet automatically after detecting a change in the preferred document format.

6. An illustrative Example

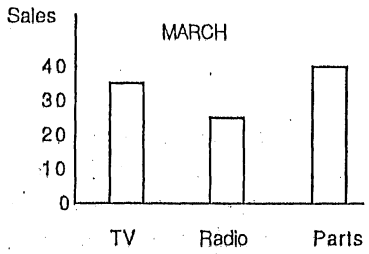
Consider a sales forecasting DSS which generates monthly sales information and presents the information in eight different formats. *Fig. 5* shows the available presentation formats (user interface elements) and *Fig. 6* shows the system usage records for the past ten months.

When the system is activated, it first analyzes the usage profile of the user to determine proper default policies and values for each operation and presentation sequence. *Fig. 7* illustrates the process by which the interface is organized. If the policy is no default, the system will present a menu (e.g. the global menu, menu *a* and menu *b* in Figure 7).

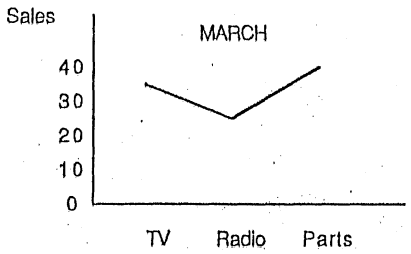
Based on the rules already discussed, the system assigns the fixed default policy to the first representation (default value is *a*-4) and the dynamic default policy to the second representation

a) Single Month Sales

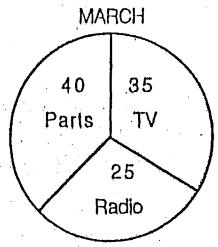
(a-1)



(a-2)



(a-3)



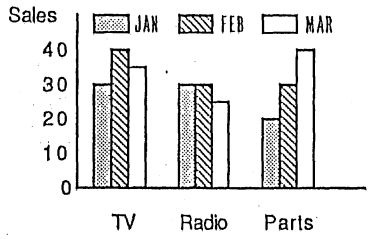
(a-4)

MONTH: MARCH

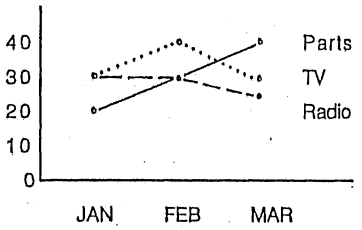
ITEMS	SALES
TV	35
RADIO	25
PARTS	40
TOTAL	100

b) Three Months Comparison

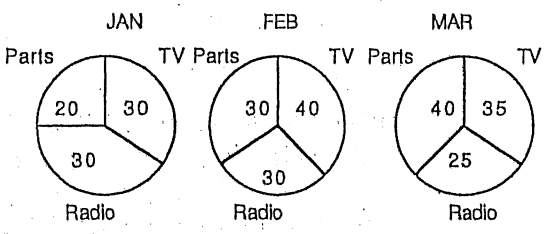
(b-1)



(b-2)



(b-3)



(b-4)

ITEMS	JAN	FEB	MAR
TV	30	40	30
RADIO	30	30	25
PARTS	20	30	40
TOTAL	80	100	100

Fig. 5. User interface elements.

month	1	2	3	4	5	6	7	8	9	10
representation 1	a-4	a-4	a-4	a-3	a-4	a-4	a-4	a-4	a-4	a-4
representation 2	b-1	b-1	b-1	b-3	b-3	b-2	b-2	b-2	b-4	b-4

Fig. 6. Usage data of the example.

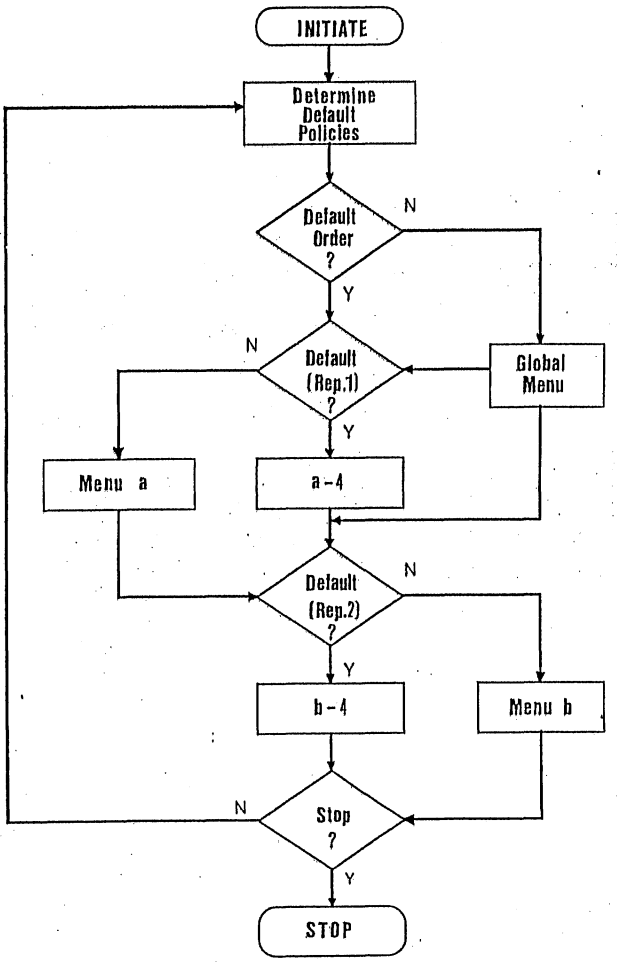


Fig. 7. Process for formulating the interface.

(default value is *b*-4). The expected number of keystrokes for the dynamic policy equals 0.3, which is the best in this case.

With regard to the sequence of presentation, previous usage records indicate a consistent pattern in which sales for a single month is first presented (series *a*) and comparison of three months' sales (series *b*) follows. Therefore, the best sequence is to adopt the fixed default policy that presents series *a* prior to series *b*.

Accordingly, the system will present information in the order of *a*-4 and *b*-4 at the eleventh month. If the user requests new formats such as *a*-3 and *b*-3, then the default *a*-4 will not be affected (because the default policy is fixed), but the default *b*-4 will be changed to *b*-3 at the twelveth month (because the default policy is dynamic).

7. Conclusion

We have presented general guidelines for user interface design and an approach to designing a self-adaptive interface. Since a user's preference changes over time, it is important for a DSS to be adaptive. The proposed mechanism requires that the system keep track of the usage profile for each user, compute the performance of various default policies based upon the usage behavior, and then determine the appropriate default for each operation and representation. Applying this mechanism to interface design involves delicate tradeoffs among the anticipated behavior, system consistency, and user control over the system.

Since we do not have a valid model for predicting human behavior, a self-adaptive interface may have limitations; e.g. users may have a strong desire to control the system and cognitively do not like a self-adaptive system. In addition, the validity of the default control mechanism may affect the value of a self-adaptive system. This indicates two promising areas for future research: developing normative user models and evaluating adaptive methods empirically.

References

[1] Alavi, M. and Henderson, J.C.: "An Evolutionary Strategy for Implementing a Decision Support System", *Management Science*, 27:11 (1981) pp. 1309–1323.

[2] Bennett, J.L.: "Analysis and Design of the User Interface for Decision Support Systems," J.L. Bennett (ed.), *Building Decision Support Systems*, Reading: Addison-Wesley (1983).

[3] Blaylock, B.K. and Rees, L.P.: "Cognitive Style and the Usefulness of Information," *Decision Sciences*, 15:1 (1984) pp. 74–91.

[4] Bournique, R. and Treu, S.: "Specification and Generation of Variable, Personalized Graphical Interfaces," *International Journal of Man–Machine Studies*, 22 (1985) pp. 663–684.

[5] Carrol, J.M. and McKendree, J.: "Interface Design Issues for Advice-giving Expert Systems," *Communications of the ACM*, 30:1 (1987) pp. 14–31.

[6] Croft, W.B.: "The Role of Context and Adaptation in User Interface Design," *International Journal of Man–Machine Studies*, 21 (1984) pp. 283–292.

[7] DeSanctis, G.: "Computer Graphics as Decision Aids: Directions for Research," *Decision Sciences*, 15:4 (1984) pp. 463–487.

[8] Edmonds, E.A.: "Adaptive Man–Computer Interface," in M.J. Coombs and J.L. Alty (eds.), *Computing Skills and the User Interface*, New York: Academic Press (1981) pp. 389–426.

[9] Edmonds, E.: "The Man–Computer Interface: A Note on Concepts and Design," *International Journal of Man–Machine Studies,* 16 (1982) pp. 231–236.

[10] Good, M.D., et al.: "Building a User-Derived Interface," *Communications of the ACM*, 27:10 (1984) pp. 1032–1043.

[11] Hagglund, S. and Tibell, R.: "Multi-style Dialogues and Control Independence in Interactive Software," in T.R.G. Green and S.J. Payne (eds.), *The Psychology of Computer Use,* New York: Academic Press (1983) pp. 171–190.

[12] Hurst, E.G., Jr.: "The Role of Humans in Decision Support Systems," Working Paper 78-02-01, Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1978.

[13] Innocent, P.R.: "Towards Self-Adaptive Interface Systems," *International Journal of Man–Machine Studies*, 16 (1982) pp. 287–299.

[14] James, E.B.: "The User Interface," *Computer Journal,* 23:1 (1980) pp. 25–28.

[15] Keen, P.G.W.: "Adaptive Design for Decision Support Systems," *Data Base*, 12:1,2 (1980) pp. 15–25.

[16] Keen, P.G.W. and Gambino, T.J.: "Building a DSS: The Mythical Man-Month Revisited," in J.L. Bennett (eds.), *Building Decision Support Systems*, Reading: Addison-Wesley (1983).

[17] Liang, T.P.: "A Self-Evolving User Interface Design for Decision Support Systems," *Proceedings of the Seventeenth Hawaii International Conference on System Sciences* (1984) pp. 548–557.

[18] Liang, T.P. and Jones, C.V.: "Design of A Self-Evolving Decision Support System," *Journal of Management Information Systems*, 4:1 (1987) pp. 59–82.

[19] Maguire, M.: "An Evaluation of Published Recommendations on the Design of Man–Computer Dialogues," *International Journal of Man–Machine Studies,* 16 (1982) pp. 237–262.

[20] Malone, T.W.: "Heuristic for Designing Enjoyable User Interface: Lessons from Computer Games," in J.C. Thomas and M.L. Schneider (eds.), *Human Factors in Computer Systems*, Norwood, N.J.: Ablex Publishing Co. (1984) pp. 1–12.

[21] Mason, M.V.: "Adaptive Command Prompting in an On line Documentation System," *International Journal of Man–Machine Studies*, 25 (1986) pp. 33–51.

[22] Norman, D.A.: "Stages and Levels in Human–Machine Interaction," *International Journal of Man–Machine Studies*, 21 (1984) pp. 365–375.

[23] Pew, R.W., et al.: *Research Needs for Human Factors,* National Academy Press, Washington, D.C. (1983).

[24] Rich, E.: *Artificial Intelligence*, New York: McGraw-Hill (1983).

[25] Rissland, E.L.: "Ingredients of Intelligent User Interfaces," *International Journal of Man–Machine Studies*, 21 (1984) pp. 377–388.

[26] Saja, A.D.: "The Cognitive Model: An Approach to Designing the Human–Computer Interface," *ACM SIGCHI Bulletin*, 16:3 (1985) pp. 36–40.

[27] Savage, R.E. and Habinek, J.K.: "A Multi-level Menu-Driven User Interface: Design and Evaluation through Simulation," in J.C. Thomas and M.L. Schneider (eds.), *Human Factors in Computer Systems*, Norwood, N.J.: Ablex Publishing Co. (1984) pp. 165–186.

[28] Sprague, R.H. and Carlson, E.D.: *Building Effective Decision Support Systems*, N.J.: Prentice-Hall (1982).

[29] Stohr, E.A. and White, N.H.: "Use Interfaces for Decision Support Systems: An Overview," *International Journal of Policy Analysis and Information Systems*, 6:4 (1982) pp. 393–423.

[30] Tyler, S.W. and Treu, S.: "Adaptive Interface Design: A Symmetric Model and A Knowledge-based Implementation," *SIGOIS Bulletin*, 7:2–3 (1986) pp. 53–60.

Appendix: An Implementation

```
/* data base of user profile */

profile(pattern_a,[1,2,3,4,2,1,3,4,3,3,3,2,2,1,3,2,2,2,2,2,3]).
profile(pattern_b,[3,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,2,2,2]).
profile(pattern_c,[2,2,1,2,2,2,3,2,2,2,2,2,1,2,2,4,2,3,2,2,1]).
profile(pattern_d,[2,1,3,4,3,1,2,4,1,4,2,3,2,1,3,3,1,4,4,2,3]).

/* Main Program */

run(User):-
    profile(User,Profile),
    analyze(Profile,Default_candidates),
    select(Default_candidates,Best),
    report(User,Best).
run(User):-
    nl,write('Sorry -- usage data not available !!'),
    nl,nl,!.

analyze(Profile,Result):-
    fixed_default(Profile,Candidacy_1),
    dynamic_default(Profile,Candidacy_2),
    append([Candidacy_1],[Candidacy_2],Result1),
    delete([],Result1,Result).

select(Default_candidates,Best):-
    list_length(Default_candidates,L),
    L=1,
    first(Best,Default_candidates).
select(Default_candidates,Best):-
    list_length(Default_candidates,L),
    L=2,
    first(Fixed,Default_candidates),
    delete_head(Fixed,Default_candidates,Default_candidates_2),
    first(Dynamic,Default_candidates_2),
    compare(Fixed,Dynamic,Best).
select([],[no_default,menu,1.0]).

compare(Fixed,Dynamic,Best):-
    third(P2,Dynamic),
    third(P1,Fixed),
    P1>P2,
    append(Dynamic,[],Best).
compare(Fixed,Dynamic,Best):-
    append(Fixed,[],Best).

report(User,[Policy, Value, Performance]):-nl,nl,
    write('-------------------------------------------------'),nl,
    write('      My suggestions for '),write(User),nl,
    write('-------------------------------------------------'),nl,nl,
    write(' The optimum default policy is <'),
    write(Policy),write('>'),nl,
    write(' Current default value is <'),
    write(Value),write('>'),nl,
    write(' The expected performance of this setting is '),nl,
    write(Performance),write(' keystrokes'),nl,nl,
```

```
    write('-------------------------------------------------'),nl,nl.

fixed_default(Profile,Candidacy_1):-
    list_length(Profile,L),L>0,
    count(Profile,L,Performance),
    candidate_1(Performance,Candidacy_1).
fixed_default(Profile,[]).

count([],_,[]).
count(Profile,L,Performance):-
    first(F1,Profile),list_length(Profile,L1),
    delete(F1,Profile,Data_1),
    list_length(Data_1,L2),
    P1 is 1 - (L1 - L2) / L,
    P1 > 0.5,!,nl,
    write('- Default policy is fixed'), nl,
    write('  Default value is '), write(F1), nl,
    write('  Performance is '), write(P1), nl,
    count(Data_1,L,Performance).

count(Profile,L,Performance):-
    first(Default_value,Profile),list_length(Profile,L1),
    delete(Default_value,Profile,Data_1),
    list_length(Data_1,L2),
    Perf is 1 - (L1 - L2) / L,nl,
    write('- Default policy is fixed'), nl,
    write('  Default value is '), write(Default_value), nl,
    write('  Performance is '), write(Perf), nl,
    append([Default_value],[Perf],Performance).

candidate_1(Performance,Candidacy_1):-
    list_length(Performance,L),!,
    L>0,
    append([fixed],Performance,Candidacy_1).
candidate_1([],[]).

dynamic_default(Profile,Candidacy_2):-
    calculate(Profile,Performance),
    candidate_2(Performance,Candidacy_2).
dynamic_default(Profile,[]).

calculate(Profile,[Default_value,Performance]):-
    list_length(Profile,L),L>0,
    keystroke(Profile,Key),last(Default_value, Profile),
    P is Key/L, nl,
    write('- Default policy is dynamic'), nl,
    write('  Default value is '), write(Default_value), nl,
    write('  Performance is '), write(P),nl,!,
    P<0.5,
    Performance is P.
calculate(_,[]).

keystroke(Profile,Key):-
    first(F1,Profile),
    delete_head(F1,Profile,Data_1),
    match_head(F1,Data_1),
    keystroke(Data_1,Key).
```

```
keystroke(Profile,Key):-
    first(F1,Profile),
    delete_head(F1,Profile,Data_1),
    not match(F1,Data_1),
    keystroke(Data_1,Key1),
    Key is Key1+1.
keystroke([],0):-!.

candidate_2(Performance,Candidacy_2):-
    list_length(Performance,L),!,
    L>0,
    append([dynamic],Performance,Candidacy_2).
candidate_2([],[]).

/* Utilities */

first(X,[X|Anything]).

last(E,List):-append(_,[E],List).

third(X,[A,B,X]).

append([],List_1,List_1).
append([A_head|Tail_1],List_2,[A_head|Tail_1_then_list_2]):-
   append(Tail_1,List_2,Tail_1_then_list_2).

delete_head(Element,[Element|Tail],Tail):-!.

delete(_,[],[]).
delete(Element,[Element|Tail],List1):-!,
   delete(Element,Tail,List1).
delete(Element,[Any|Tail1],[Any|Tail2]):-
   delete(Element,Tail1,Tail2).

list_length([],0).
list_length([X|Tail],Length):-list_length(Tail,K),
   Length is K+1.

match_head(Element,[Element|Anything]).
```