A Method for Consistent Design of User Interaction with Multifunction Devices

Dong San Kim and Wan Chul Yoon

Department of Industrial & Systems Engineering,
Korea Advanced Institute of Science and Technology, 373-1,
Gu-seong, Yu-seong, Daejeon, Korea
kimdongsan@gmail.com, wcyoon@kaist.ac.kr

Abstract. Over the last decade, feature creep and the convergence of multiple devices have increased the complexity of both design and use. One way to reduce the complexity of a device without sacrificing its features is to design the UI consistently. However, designing consistent user interface of a multifunction device often becomes a formidable task, especially when the logical interaction is concerned. This paper presents a systematic method for consistent design of user interaction, called CUID (Consistent User Interaction Design), and validates its usefulness through a case study. CUID, focusing on ensuring consistency of logical interaction rather than physical or visual interfaces, employs a constraint-based interactive approach. It strives for consistency as the main goal, but also considers efficiency and safety of use. CUID will reduce the cognitive complexity of the task of interaction design to help produce devices that are easier to learn and use.

Keywords: Interaction design method, consistency, constraint-based design.

1 Introduction

Over the last decade, interactive devices such as mobile phones have become increasingly complicated despite no little efforts have been invested in usability engineering than ever before. One of the main reasons for this is *creeping featurism or feature creep*: the tendency for the number of features in a product to rise with each release of the product [1-3]. To reduce the negative effects of feature creep (e.g., complexity of both design and use), some researchers and companies emphasize the "simplicity" of a product [4-5]. In the field, however, it is often the case that removing a feature is much more difficult than adding it [3]. The primary reason for this is that consumers want products with more features even if they know that they will probably never use most of the features and products with more features will lead to decreased usability [6-8]. While most people desire more features, at the same time they also want ease of use. In general, however, the more features a product has, the higher the complexity of using the product is [7]. For this reason, many manufacturing companies are faced with a dilemma: should they make their products more capable or should they make them more usable [6-7].

M. Kurosu (Ed.): Human Centered Design, HCII 2009, LNCS 5619, pp. 202–211, 2009.

One way to resolve this dilemma – that is, to reduce the complexity of a device without sacrificing its features – is to design the device consistently. If a user interface (UI) is designed consistently, the user will benefit from what psychologists call *transfer of training*. In other words, learning to do one thing in one context will make it easier to learn how to do similar things in similar contexts [9]. However, it is often not easy to design the user interface of a multifunction device consistently for several reasons: (1) it is very cognitively demanding and error-prone, because the designer should consider hundreds of design variables and relations between them; (2) the size of the UI and the number of the given interface controls such as push buttons are getting smaller; (3) the product development environment has become fast-paced due to increasing global competition; (4) many designers collaborate on the design; and (5) product requirements typically change often during the design process.

Therefore, there is a strong need for a well-developed method that supports complex design processes. Although the importance of consistency has widely been noted and many formal models for evaluating interface consistency have been developed in the HCI literature, relatively few studies have been carried out on methods or tools for supporting the consistent design of user interaction, particularly interaction for multifunction or convergence devices.

This paper introduces a method for designing user interaction consistently, called CUID (Consistent User Interaction Design). It focuses on ensuring consistency of *logical interaction* – task procedures, operation or function availabilities in system states, and interface controls (or interaction methods) for each available operation – rather than *physical or visual interface elements* such as colors, sizes, locations, etc. In the next section, we briefly review previous research on consistency in UI design. Design problems that are dealt with in CUID and main approaches to address the problems are discussed in Section 3. In Section 4, a case study for validating CUID is presented. Finally, Section 5 concludes the paper and discusses directions for future work.

2 Related Work

Consistency is one of the most important principles in user interface design. The first rule of Ben Shneiderman's well-known eight golden rules of interface design is "Strive for consistency" [10]. Although clearly defining consistency is not simple [11], it can be loosely defined as "*do similar things in similar ways"* [14]. According to previous studies, consistency may sometimes conflict with other design principles and it is sometimes beneficial to be inconsistent [11-13]. However, it is generally agreed that consistently designed systems are easier to learn and easier to remember, reducing performance time and errors, and as a result increasing user satisfaction as compared to inconsistently designed systems [15].

Previous studies have developed a number of methods for ensuring interface consistency in three different areas: design guidelines, models or tools for evaluating interface consistency, and tools for generating consistent user interfaces. Many kinds of guidelines and standards for interface consistency have been developed thus far. Presently, most commercial applications have their own interface guidelines and most of the guidelines (e.g., [16]) include consistency as one of the major principles. Ozok

and Salvendy [17] suggested twenty guidelines for the design of consistent Web-based interfaces. Examples are: *"Use same words for same items, rather than using synonyms"; "Be consistent in terms of character sizes, use of upper/lower case letters, spacing, punctuation, character colors, and wording in your text."* With these design guidelines, however, consistent design of user interaction with a complicated device is not easy. Compliance with guidelines or standards is not sufficient to ensure consistency, because the guidelines and standards leave a fair amount of leeway for the designers [15].

Therefore, there has always been much need for developing models and tools that can evaluate interface consistency. Since the 1980s, several formal models such as TAG [18], APT [19], and PROCOPE [20] have been developed for evaluating the consistency of designed user interfaces. In these models, task procedures are represented by rules, and fewer rules indicates higher consistency. Formal models, however, require much time to learn and use grammar in a fast-paced environment. In the 1990s, a few software tools for evaluating interface consistency were introduced. GLEAN [21] makes predictions about human performance on a user interface based on the GOMSL model, which can predict transfer times between tasks and identify consistency problems between similar tasks. However, it has at least two limitations. First, GOMSL regards two task goals as similar when they are different on only one of the verb and the noun in the given statements. Semantic similarity between two task goals with both different verbs and nouns is not considered (e.g. "increase the volume" and "enlarge the photo"). Second, when counting the number of similar statements between two procedures, it stops counting when it encounters two statements that are non-identical even if the remaining statements are the same. SHERLOCK [22], a family of consistency analysis tools, evaluates visual and textual properties of user interfaces. It was designed to detect inconsistencies in terminology, capitalization, typefaces, colors, button sizes, placements, etc. As it focuses only on visual and textual consistency, consistency of logical interaction is not considered.

In recent years, tools for automatically generating consistent user interfaces have also been developed. SUPPLE [23] automatically generates layouts for user interfaces using an optimization approach to choose controls and their arrangement. A branch-and-bound constrained search algorithm is used for mapping between a set of interface elements – units of information that need to be conveyed via the interface between the user and the controlled appliance or application – and available UI widgets (e.g., spinner, tab pane, slider). However, it also addresses only the design of physical or visual interfaces. UNIFORM [24] automatically generates remote control interfaces that are personally consistent, which means that the interfaces generated for each user are consistent with that particular user's past interface experiences. It can automatically identify similarities between different devices and users may specify additional similarities. The similarity information allows the interface generator to use the same type of controls for similar functions, place similar functions so that they can be found with the same navigation steps, and create interfaces that have a similar visual appearance. However, as the authors noted, its solutions are often limited because of a lack of useful semantic information about unique functions. Moreover, it takes a substantial amount of time (about 5 hours for a VCR and one week for three different VCRs for even an expert) to complete a specification for an appliance, on which similarities between different appliances are based.

3 CUID Problems and Approaches

As shown in Fig. 1, CUID deals with three kinds of problems in user interaction design, each of which can be considered as matching between two adjacent abstraction levels:

1) *Task-Procedure design*: the design of procedure(s) of each task
2) *State-Operation design*: the design of operation[1] or function availability in each system state and of the state change by each available operation
3) *Operation-Control mapping*: the mapping between available operations and interface controls (or interaction methods)

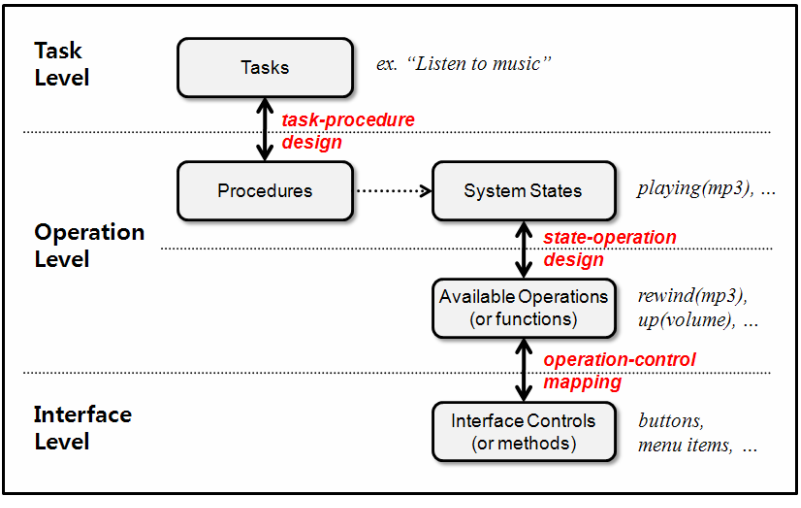

Fig. 1. Three main design problems in CUID

The three design problems pertain to the logical consistency rather than the physical or visual consistency of UIs. Among them, the first problem concerning the consistency of task procedures has been extensively studied in the HCI discipline; GOMS and TAG are representative models that address the problem. The other two design problems, however, have not received much attention in the HCI literature. Although these problems are not significant in designing a relatively simple system with a small number of features, they become important issues when designing a convergence or multifunction device.

In solving each design problem, consistency is used as the primary design principle. In this paper, the term consistency is defined more broadly than in most previous studies: *keep a relation between design elements the same at all levels*. It is beyond "do similar things in similar ways"; it deals with not only synonyms – similar things – but also antonyms – opposite pairs of things such as left and right. Therefore, for each

[1] In this paper, an operation represents a logical action (e.g. "delete a file") rather than a physical action (e.g. "press a push button").

design problem, at least two principles are used. For example, in operation-control mapping, the principles are "assign the same or similar interface controls to similar operations" and "assign each of a pair of controls to each of a pair of operations".

Although consistency, the primary goal of CUID, is generally desired, it is obviously not the only desirable usability characteristic. Taking consistency only as the guiding rule in design can be very risky [10-12],[14]. Therefore, CUID strives for consistency as the main goal, but it also tries to satisfy other usability characteristics such as efficiency and safety by applying the following rules:

- Frequently performed tasks should be provided with shortcut procedures (for efficiency).
- Urgent tasks, quick start of which is required, should be provided with shortcut procedures (for efficiency).
- If necessary, more than two procedures for a task or interaction methods for an operation should be possible (for efficiency and learnability).
- Critical tasks, the wrong performance of which can be critical, should include a confirmation step in the procedures (for errors/safety).

To support a consistent design of interaction, CUID uses two approaches: constraint-based design and highway-based design, which will be explained below.

3.1 Constraint-Based Design

Constraint-based design is a design approach that is widely used in engineering design [25]. In a constraint-based approach, a design problem entails determining the values of design variables while satisfying a set of constraints that are tightly coupled and sometimes in conflict with one another, and a constraint is generally defined as the bound on a single design variable or the relation among a set of design variables. Since variables are typically interconnected via constraints, a design problem constructs a network of variables and constraints known as a constraint network [26]. When a value is assigned to a variable, the design decision is propagated through the network, influencing the decisions of values of other connected variables. Constraints are generally viewed as limiting factors in design. In constraint-based design, however, constraints are not considered as negative factors but as *design drivers* that drive the exploration of the solution space [27].

To date, only a few studies have applied constraint-based approaches to UI design (e.g.,[23],[28]). A constraint-based approach is utilized for CUID because a key feature of the cognitive processes of UI designers is that they are constraint-based. In best UI design practices, designers put almost certain solutions into the partial solution set as soon as they are found during the process and use them as design constraints to narrow down solutions for the adjacent problems. Since the certain solutions are scattered in the design space, the designer will leap around problems, collecting solutions in an opportunistic manner [29].

Fig. 2 illustrates the present constraint-based design. Design variables and possible values at each abstraction level – tasks, procedures, states, operations, and interface controls – form their own network by relations among them. When the value of a design variable is determined (drawn in the solid bidirectional arrows in Fig. 2), the design decision becomes a constraint that influences the decisions of values of other

variables that are connected to this variable (drawn in the dotted bidirectional arrows in Fig. 2). This constraint-based matching between two adjacent levels forms three kinds of representations: a parallelogram, triangle, and inverted triangle.

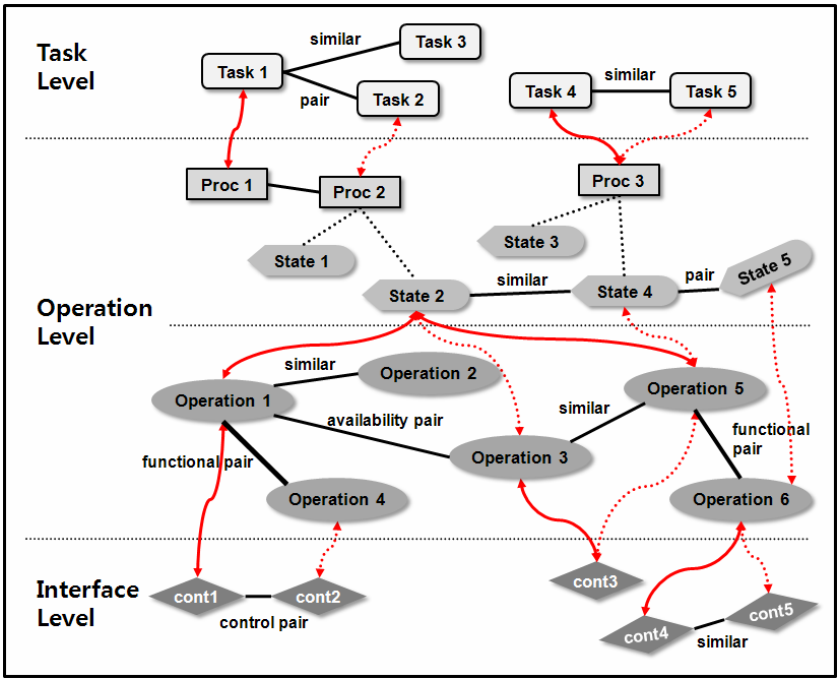

Fig. 2. An illustration of the constraint-based approach of CUID

Note that design constraints are divided into horizontal or within-level constraints – relations among design variables within the same abstraction level – and vertical or between-level constraints – relations among variables and values in different abstraction levels (decisions made by the designer). Table 1 summarizes the constraints that are used for each design problem.

Vertical constraints, i.e., variable-value relations, include both relations that are predefined and that are determined while designing. The predefined relations are external constraints for consistency with other devices and with users' mental models. These are social standards or cultural conventions. Procedure templates for each task type, essential operations that should be available in all or almost all system states, and required mappings between available operations and interface controls (e.g., "play(x)" operation and "▶" button) are used as the external constraints. In this respect, CUID considers both external and internal consistency. Horizontal constraints, i.e., within-level relations, involve *groups* – that are formed by semantic or functional similarity – and *opposite pairs* that are semantically or functionally antonymous to each other. To assess the semantic similarity among variables in each design problem, taxonomies of user tasks, system states, and user operations have been developed.

Table 1. Constraints used for each design problem in CUID

Design problem	Vertical constraints (between-level)	Horizontal constraints (within-level)
Task-Procedure design	Procedure templates Task-procedure decisions	Task groups based on types Opposite pairs of tasks
State-Operation design	Essential operations State-operation decisions	State groups based on types Opposite pairs of states
Operation-Control mapping	Required mappings Operation-control decisions	Operation groups based on types Opposite pairs of operations
		Control groups based on similarity Opposite pairs of controls

The constraint-based approach would improve not only the efficiency of UI design when a large number of design variables and relations among them should be considered, but also the usability of the generated UI by reducing the important elements that designers are likely to overlook.

3.2 Highway-Based Design

In a constraint-based design process, design results and time required can vary depending on the order in which design variables are assigned to values. In CUID, to address this problem, highway variables – representatives of the variables – are selected. The values of the highway variables are first determined, and then the values of the other variables are assigned by using the decisions as highway constraints. Highway tasks are selected by considering the frequency and generality of tasks, and highway states are selected among the states that belong to highway tasks. Operations available in highway states are defined as highway operations.

4 CUID Design Process

Fig. 3 shows the CUID design process. It is comprised of three main phases – task-procedure design, state-operation design, and operation-control mapping – and each design phase is composed of three or four steps. There are feedback loops among the three phases and among the steps in a phase, and thus the overall process proceeds iteratively.

Each phase begins with an analysis of its own design variables, in which design constraints, relations among the variables are determined. Relations among tasks (i.e. task groups determined by task types and opposite pairs of tasks) are elicited by task analysis, relations among system states (state groups by state types and opposite pairs of states) by state analysis, and relations among operations (operation groups by operation types and opposite pairs of operations) and relations among interface controls (control groups and opposite pairs of controls) by operation and control analysis.

After the constraints to be used in a phase are identified, the main step of the phase – decisions of values of the design variables – follows. In these steps, procedures of the given tasks, available operations in each system state and the state change by each available operation, and mappings between operations and interface controls are

determined. For each group of variables, the values of highway variables are first determined, and the values of the other variables are then assigned. The value of a variable is determined by considering the previous decisions of the values of other variables – including highway variables – that are similar or opposite to the variable. Each of the taxonomies, by which the groups of variables are determined, has two or three levels so that multi-level groups can be formed. The designer can extend or reduce the range of similar variables by changing the abstraction level.

In the last step of each phase, design decisions made in the phase are evaluated and refined. For example, the designer can assign an appropriate operation (or function) to a remaining interface control – to which no operation is assigned in a system state – by considering which operations are assigned to the control in other states. At these "refinement" steps, the designer can check if he or she missed some relations among design variables in the main steps.

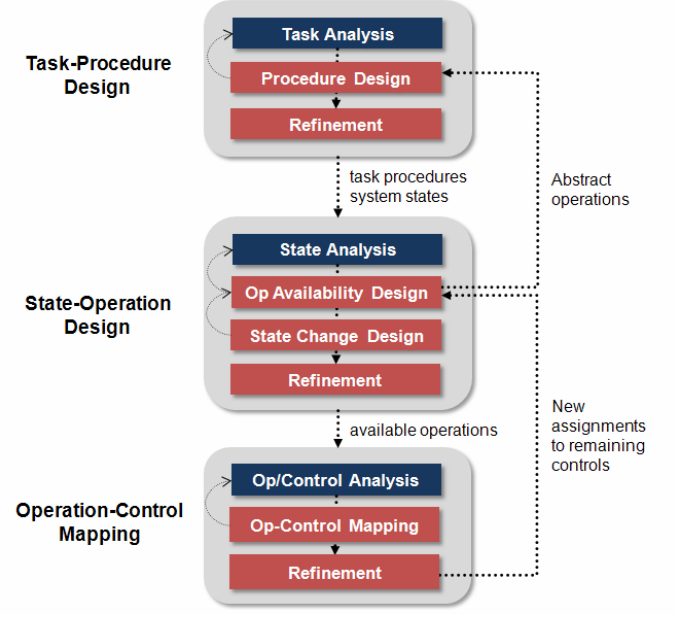

Fig. 3. The CUID design process

5 Case Study

To evaluate the usefulness of CUID, we first developed a prototype software tool that supports the CUID design process, and then the following procedure was applied: (1) the interaction part of the experimental device, a recently commercialized multifunction mp3 player, was redesigned with CUID; (2) logical inconsistency problems the device has were identified by a heuristic evaluation with six users; and (3) how many inconsistencies were resolved in the redesigned interaction was checked. Twenty-three inconsistency problems in total were collected from the six users. We found that

21 of the 23 inconsistency problems (91%) did not exist in the redesigned interaction. This indicates that CUID can be a useful tool for consistent design of user interaction.

6 Conclusions and Future Work

In this study, a method for consistent design of user interaction, called CUID, was developed. To support consistent design, CUID employs a constraint-based approach. Vertical or between-level constraints, meaning the relations between design variables and values, and horizontal or within-level constraints, meaning the relations among variables or values within an abstraction level, are both considered. CUID also uses a highway-based approach in which highway variables, representatives of variables, are selected; the values of the highway variables are first determined; and the values of the other variables are then assigned by using the decisions on the highway variables. CUID strives for consistency as the main goal, but it also considers other usability characteristics such as efficiency and errors/safety.

There is still much research to be done in the area of interface consistency, which has seen little activity in the past decade [24]. In particular, consistency of logical interaction rather than that of physical or visual interfaces needs to be addressed. Future work on CUID will be directed toward the following goals. First, CUID will be refined by redesigning user interaction for various kinds of multifunction devices such as mobile phones and multifunction printers. Second, the taxonomies of tasks, system states, and operations should also be refined, as each of the current taxonomies does not cover all possible types. Third, a consistency measure will be developed to assess the level of consistency of user interaction quantitatively. This will enable the optimization of interaction design. It is expected that CUID, for UI designers, will reduce the complexity of designing user interaction for an interactive device with many functions, and for users it will contribute to providing them with products that are consistently designed, and thus easier to learn and use.

References

1. Norman, D.A.: The Psychology of Everyday Things. Basic Books, New York (1988)
2. Lee, D.S., Woods, D.D., Kidwell, D.: Escape from Designers' Dilemma on Creeping Featurism. In: Proceedings of Human Factors and Ergonomics Society Annual Meeting, Test and Evaluation, pp. 2562–2566 (2006)
3. Surowiecki, J.: Feature Presentation. The New Yorker, May 28 (2007)
4. Philips: Our Brand Promise Sense and Simplicity, http://www.philips.com/about/brand/brandpromise
5. Physorg: Vodafone Simply: The More The Better? May 21 (2005), http://www.physorg.com/news4197.html
6. Rust, R.T., Thompson, D.V., Hamilton, R.W.: Defeating Feature Fatigue. Harvard Business Review 84(2), 98–107 (2006)
7. Norman, D.A.: Simplicity is Highly Overrated. Interactions 14(2), 40–41 (2007)
8. Norman, D.A.: Simplicity is Not the Answer. Interactions 15(5), 45–46 (2008)
9. Monk, A.: Noddy's Guide to Consistency. Interfaces 45, 4–7 (2000)

10. Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-Computer Interaction, 3rd edn. Addison-Wesley, Reading (1998)
11. Grudin, J.: The Case Against User Interface Consistency. Communications of the ACM 1(1), 59–71 (1989)
12. Grudin, J.: Consistency, Standards, and Formal Approaches to Interface Development and Evaluation. ACM Transactions on Information Systems 10(1), 103–111 (1992)
13. Ketola, P., Hjelmeroos, H., Raiha, K.J.: Coping with Consistency under Multiple Design Constraints: The Case of the Nokia 9000 WWW Browser. Personal Technologies 4, 86–95 (2000)
14. Reisner, P.: What is Consistency? In: INTERACT 1990, pp. 175–181 (1990)
15. Nielsen, J.: Usability Engineering. Academic Press, Boston (1993)
16. Apple: iPhone Human Interface Guidelines. Apple Inc. (2007)
17. Ozok, A.A., Salvendy, G.: Twenty Guidelines for the Design of Web-based Interfaces with Consistent Language. Computers in Human Behavior 20, 149–161 (2004)
18. Payne, S.J., Green, T.R.G.: Task-Action Grammar: A Model of the Mental Representation of Task Languages. Human-Computer Interaction 2, 93–133 (1986)
19. Reisner, P.: APT: A Description of User Interface Inconsistency. Int. J. of Man-Machine Studies 39, 215–236 (1993)
20. Poitrenaud, S.: The PROCOPE Semantic Network: An Alternative to Action Grammars. Int. J. of Human-Computer Studies 42, 31–69 (1995)
21. Kieras, D.E., Wood, S.D., Abotel, K., Hornof, A.: GLEAN: A Computer-Based Tool for Rapid GOMS Model Usability Evaluation of User Interface Designs. In: Proceedings of UIST, pp. 91–100 (1995)
22. Mahajan, R., Shneiderman, B.: Visual Consistency Checking Tools for Graphical User Interfaces. IEEE Transactions on Software Engineering 23(11), 722–735 (1997)
23. Gajos, K., Weld, D.S.: SUPPLE: Automatically Generating User Interfaces. In: Proceedings of IUI, pp. 93–100 (2004)
24. Nichols, J., Myers, B.A., Rothrock, B.: UNIFORM: Automatically Generating Consistent Remote Control User Interface. In: Proceedings of CHI, pp. 611–620 (2006)
25. Lin, L., Chen, L.C.: Constraints Modelling in Product Design. J. of Eng. Design 13(3), 205–214 (2002)
26. Hashemian, M., Gu, P.: A Constraint-Based System for Product Design. Concurrent Engineering: Research and Applications 3(3), 177–186 (1995)
27. Kilian, A.: Design Innovation Through Constraint Modeling. Int. J. of Architectural Computing 4(1), 87–105 (2006)
28. Borning, A., Duisberg, R.: Constraint-Based Tools for Building User Interfaces. ACM Transactions on Graphics 5(4), 345–374 (1986)
29. Yoon, W.C.: Task-Interface Matching: How We May Design User Interfaces. In: Proceedings of the 15th International Ergonomics Association Triennial Congress (2003)