Adaptation in Automated User-Interface Design

Jacob Eisenstein and Angel Puerta

RedWhale Software
277 Town and Country
Palo Alto, CA 94305 USA
+1 650 321 3437
{jacob, puerta}@redwhale.com

ABSTRACT

Design problems involve issues of stylistic preference and flexible standards of success; human designers often proceed by intuition and are unaware of following any strict rule-based procedures. These features make design tasks especially difficult to automate. Adaptation is proposed as a means to overcome these challenges. We describe a system that applies an adaptive algorithm to automated user interface design within the framework of the MOBI-D (Model-Based Interface Designer) interface development environment. Preliminary experiments indicate that adaptation improves the performance of the automated user interface design system.

Keywords

Model-based interface development, machine learning, decision trees, theory refinement, user interface development tools, interface models, theory refinement

INTRODUCTION

Design is a human ability that eludes formalization. Human designers are often not aware of following strict rule-based procedures when they make design decisions. Moreover, design decisions often cannot be evaluated on any kind of objective scale of utility. Two qualified human experts may make strikingly different decisions, and still there may be no way to identify one expert's decision as unambiguously "correct." Even though it is clear that in such a case neither expert is wrong, it is also clear that not just any decision would do. Although there may be more than one right answer to a design problem, there are still wrong answers. It is this condition that makes design a creative task, and this is why design proves so challenging for computer automation.

We are specifically interested in automating particular features of user interface design. Previous work in the automation of user interface design has had mixed success. Researchers have developed systems that are effective for narrowly focused domains including, for example, automatic generation of forms, or automatic generation of dialog boxes for database access [9, 3]. However, no technique has been shown to be applicable at a general level. We claim that adaptive, automated systems that solicit and respond to user feedback may be able to succeed where previous efforts have failed.

An adaptive system for automated user interface design will benefit both designers and interface-design researchers. Designers will benefit in at least three ways:

1. User-interface design software will adapt to accommodate their stylistic preferences. In the case of individual idiosyncrasies, designers can trust that the software will take their preferences into account. Where there are whole schools of thought on design—e.g., within a single software company—adapted versions of the interface design software can be distributed.

2. Designers will find it easier to explain their stylistic preferences to others, since the adaptive algorithm will extract a formal description of that style.

3. Technological developments in user-interface design can be accommodated by existing design software without the need for updates or patches. If, for example, a new user-interface widget is introduced, the automatic design algorithm can learn to handle it by observing the designer's behavior.

Adaptation will also benefit researchers, who can discover new information about the way designers make decisions, by observing the results of using the adaptive algorithm. Errors in the automatic design formalism will be easily identified and corrected. Differences between schools of design will be made explicit. Researchers will discover what design knowledge is consensus and what is a matter of preference. In sum, adaptation will serve as a formal

methodology that will help researchers to develop and refine general aspects of a theory of user interface design.

SURF – OUR GUIDING PHILOSOPHY

Our work is guided by four main principles which distinguish it from other forays into automatic user-interface design. These principles are represented by the acronym "SURF."

1. **SENSITIVITY**. It is important that the interface designer is aware that the adaptive algorithm is sensitive to what the designer is doing. The adaptive algorithm ought to respond to a minimum of feedback from the designer. This will make it easier for the designer to become familiar with the adaptive algorithm, and the designer will be more likely to trust the interface-design environment as a whole. If the adaptive algorithm is not sufficiently sensitive to user feedback, the designer will have a difficult time predicting the behavior of the adaptive algorithm, and may find it more cumbersome and confusing than useful. In terms of the learning algorithm we will employ, this consideration leads us to favor local learning methods over large-scale batch learning.

2. **UNDERSTANDABILITY**. In any automated system where a symbiotic, cooperative relationship with the user is desired, it is important that the user of the system understands how and why automatic decisions are made. The automation algorithm that generates the user interface must be comprehensible to the interface designer, and the designer must be able to alter it explicitly and directly, if so desired. Symbolic methods are therefore favored over less user-comprehensible strategies, such as neural networks.

3. **REFINEMENT**. In developing the adaptive algorithm, we favor *theory refinement*, rather than learning from scratch. There is no need for our adaptive algorithm to automatically build a knowledge base of automatic user interface design rules from the ground up. There already exists a large body of work on the automation of user interface design decisions. We have designed our adaptive algorithm to take full advantage of this body of work. This principle supports the principle of sensitivity because it is easier to build an adaptive system that is sensitive when given a head start in the form of an existing base of knowledge.

4. **FOCUS**. We do not believe that at the present time it is reasonable to expect to move directly from an abstract description of a user interface to the automatic creation of the complete user interface. Rather, we want to apply automation only to those features of the design process where we think automation has something to offer. We have focused our work on the selection of interactors—visual elements, such as buttons or sliders, that allow the user to view or manipulate data. Our automation algorithm returns an ordered list of interactors; the interface designer then chooses from this list while performing layout. It is hoped that the number of design decisions that can be automated will increase with further research. For the present, the only way to build a system that is usable in real-world design projects is to focus on those areas of design that are particularly amenable to automation and leave other areas of design in human hands.

In summary, we will describe an adaptive, automated system for user interface design that adheres to the principles of SURF. This system is part of a larger framework, which we will now describe.

FRAMEWORK

MOBI-D

The adaptive algorithm described in this paper supplements **MOBI-D**, an existing model-based interface development environment [6]. Model-based systems for user interface development require the specification of a declarative interface model that explicitly describes all relevant aspects of the user interface in a formal language. MOBI-D then provides a comprehensive suite of tools to aid in the development and refinement of the interface model.

A range of tools is provided in order to handle each stage in the interface development cycle. First, a knowledge elicitation system called **U-TEL** helps the user of the interface develop models of the interface's data and task structures [11]. Next, the interface designer uses **model editors** to create relations between the more abstract elements in the data and task structures and the more concrete elements that describe the actual look and feel of the interface. For the final stage of development, we provide **MOBILE**, a layout tool that can be configured to reflect the decisions made at previous stages in the design process. [7]

Interface design is viewed theoretically as a process of creating mappings between various formal elements at different levels of abstraction. For example, an abstract task object may map on to abstract domain object that is manipulated when the user executes that task. But that same domain object may map on to a concrete interactor, such as a checkbox, through which the user actually performs the manipulation. **TIMM**, The Interface Model Mapper, is a tool within MOBI-D that assists designers in the generation of these mappings [8]. Some of these mappings—the principle of Focus forbids us from saying that “all”—can be made automatically. In this paper, we describe how TIMM automatically generates mappings between domain objects

and concrete interactors, and how this automatic generation of mappings benefits from adaptation.

Decision Trees

In order to perform the automatic mappings, a decision tree is used. A decision tree defines a procedure for classifying cases into groups based on discriminants. Discriminants are features of the cases that may be relevant to how they are sorted. The decision tree specifies which discriminants to consider and in what order. **Figure 1** gives a simplified example of a decision tree for interactor selection.

Consider the following example. Suppose a hi-tech movie theater wanted to develop a system to recommend current films to prospective moviegoers. The first discriminant is the viewer's age; movies that have been rated 'R' are restricted to adults, and should not be recommended if the viewer is under 18. Likewise, children's movies are generally not to be recommended to viewers over the age of 13. Now consider the group that falls between the two boundaries: viewers between the ages of 13 and 18. For this group, another discriminant is applied to determine whether or not the viewer is on a date (with a boyfriend or girlfriend). If so, then the selection is narrowed down to films that might be appropriate for a date, and then a third and final discriminant can be applied to determine the viewer's favorite actor or actress. By this method, the prospective moviegoer is classified into a viewer group with similar interests, and a short, customized list of recommendations is generated.

A decision tree offers two significant advantages as a method for automation. First, decision trees are extremely readable: more so than knowledge-bases of rules, and certainly more so than other methods such as neural or Bayesian networks [14]. Rule-based systems are difficult to read because they are sensitive to the order in which the rules are applied. Often, one must analyze a whole series of rules in order to be able to predict the effect. Bayesian methods require the user to consider a large number of relevance values; again, it is difficult to predict the effect of changes to a large-scale Bayesian network. Neural networks offer little in the way of comprehensible justification for the effects they produce. By contrast, the structure of a decision tree makes it easy to predict its effects. The second advantage offered by decision trees is that there is already a body of work on applying decision trees to the automatic selection of interactors. Vanderdonckt has created a comprehensive decision tree based on interface-design guidelines given by a wide variety of sources [13]. By using decision trees, we are able to take advantage of this work.

The decision tree in our movie example used certain discriminants, such as age and favorite actor, to classify moviegoers into consumer groups. In this application, domain elements—objects that the user of the interface will modify or view—are classified, and are assigned a set of possible interactors based on this classification. In essence, mappings are created between domain objects and

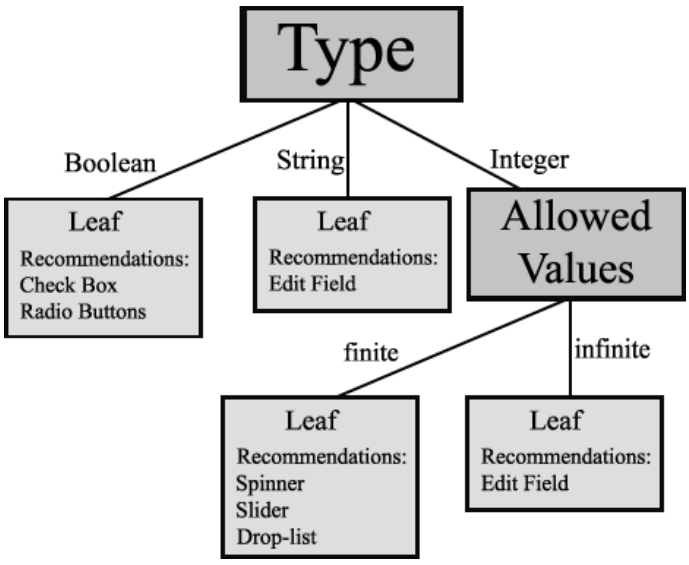

Figure 1: A Simple Decision Tree for Interactor Selection

interactors. We use the following discriminants for user interface design:

1. **Type**. e.g. Boolean, integer, float. Assigning an edit field to a domain element whose type is Boolean makes little sense, as it would require the user to manually type in "true" or "false." A Boolean element is best mapped to a check box or radio button.

2. **Number of Allowed Values**. When there are only a few allowed values, an interactor such as a set of radio buttons may suffice. But when there are many allowed values, a list box is better. When there so many allowed values that even a list box cannot fit in the dialog window, a drop-list is used.

3. **Number of Sibling Domain Elements**. This is a measure of how many other pieces of the domain model are likely to be modified on the same screen. When screen crowdedness may become an issue, the more space-efficient interactors are favored.

4. **Range**. Information about whether a domain object is a range of continuous data or a set of discrete allowed values affects interactor selection. For example, a domain element that consists of a deep range of integers will be better handled by an edit field and spinner than by a list box of allowed values.

5. **Similarity of Near Values**. If near values in the range are similar, then a slider might be helpful. But if near values bear no similarity—as is the

case in most strings, for example—then a text box would be the only reasonable choice.

We claim that this set of discriminants, while clearly not exhaustive, is broad enough to support automation.

THE ADAPTATION ALGORITHM

TIMM is the tool in the MOBI-D suite that assists the interface designer in the selection of interactors [8]. The goal of automation is to produce an ordered list of recommendations for the interface designer. But the interface designer always has the option of correcting the automatic interactor recommendations made by TIMM. Whenever the interface designer corrects one of the suggestions made by TIMM, the interaction is recorded as an *error*. Otherwise, the recommendation is considered successful. After a session is finished, TIMM applies an adaptive algorithm to correct its decision tree, based on the entire history of cases.

The adaptive algorithm has three operators for altering the decision tree. Whichever operator can reduce the number of errors by the greatest amount is selected. Operators are applied sequentially until no operator can reduce the total number of errors in the history of cases. The operators are as follows:

1. Change the recommended interactors for a given leaf of the tree.
2. Alter the boundary conditions for a branch.
3. Add a branch, and then set the output of the new leaves.

Changing the recommendations for a given leaf is the simplest operation. Suppose that the decision tree dictates that Boolean domain elements are best treated by a checkbox, but the designer selects radio buttons instead. This stylistic preference can be handled by changing the recommendation of the leaf for Boolean domain variables to radio buttons.

An alteration of the boundary conditions is more complex. In our movie recommendation example, the age 13 is a boundary condition, because it draws a line between two groups of moviegoers: children and teens. Now imagine that market research discovers that young adolescents have become more sophisticated (or perhaps jaded), and that the age 11 would be a better cutoff. This would be an alteration of boundary conditions. The same sort of thing can occur with interactor selection. Suppose that the decision tree initially uses a boundary of seven interactors per dialog in order to determine whether the screen is crowded. If there are more than seven interactors, then the interactor recommendations will reflect the need to conserve screen space; otherwise, screen space conservation is not considered. But if the interface designer favors unusually small dialog windows, this boundary might not be appropriate. It might be necessary to conserve space if there are more than five interactors in a single dialog. This is the kind of situation that would necessitate a shift in boundary conditions.

```
make_initial_recommendations();
record_user_selections();
loop {
      make_recommendations();
      count_errors();
      if (errors = 0) break;
      find_best_operation();
      if (error_gain > min_threshold){
            apply_operation();
            next;
      }
      else break;
}
```

Figure 2: The Adaptation Algorithm

The addition of a branch to the decision tree is necessary when there is a relevant piece of information that the decision tree did not consider. For example, suppose that the original decision tree recommends edit fields for all domain elements whose type is "string," irrespective of the number of allowed values. However, the user interface designer might take the number of allowed values into account, favoring list boxes when the number of allowed values is finite. The only way to correct the decision tree so that it will accommodate the designer's preference is to add a branch to take this discriminant into account.

ADAPTATION IN ACTION

Experiment 1

The adaptation algorithm was first applied to a small-scale application interface for a hypothetical program: "The Multi-User Banking Information Console." This is a small-scale interface, with 22 interactors. It is designed to allow a variety of users to view and control the money in a bank. Users range from clients to tellers to the CEO. In this experiment, an interface designer modified the automatically generated suggestions. The designer was constrained by certain features specific to the application, and had stylistic preferences that were different from the those reflected in the decision tree. The designer reviewed the automated suggestions, and then modified them while listing reasons for the modification. The adaptation algorithm was then applied to alter the decision tree so as to better conform to the designer's preferences.

The automation algorithm returns an ordered list of recommended interactors, as shown on the screen shot in **figure 3**. Errors are weighted: failure to recommend an interactor that the designer felt was necessary was the greatest error, recommending an interactor that the designer considered unnecessary was of secondary importance, and an incorrect ordering of recommendations was the least important error. Based on these weights, an overall error

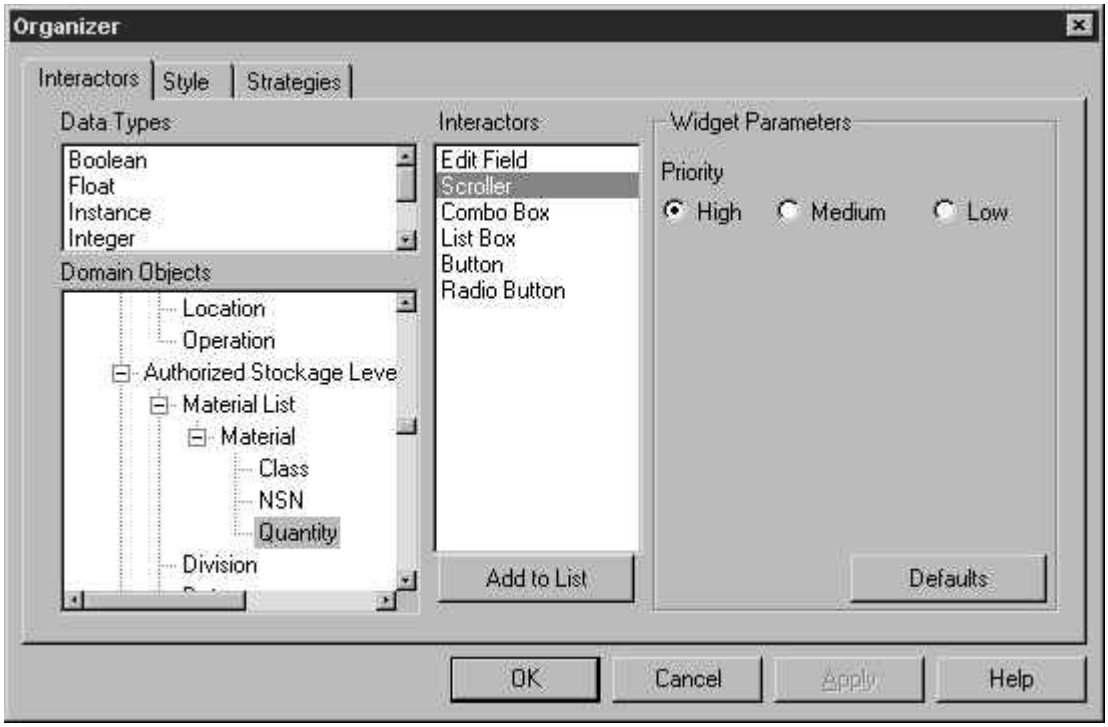

Figure 3: Screenshot of TIMM with ordered list of interactors

metric was developed that took into account the number of errors and their severity. On this scale, the adaptation algorithm was able to correct for 98.8% of the errors originally indicated by the designer. Had the designer not deliberately been slightly inconsistent, the algorithm would have been able to correct for all of the error.

Of all of the errors committed by the original decision tree, 63% represent differences of opinion on matters of style. A few stylistic questions came up many times, yielding the high count. 24% of the errors were caused by the incompleteness of the decision tree. Thus, we can expect 87% of the error correction in this case to be applicable and useful for future projects by this interface designer. The remaining 13% of the error correction involved compensating for certain specific features of this particular interface.

The experiment was deliberately designed to include constraints that could not be accounted for by TIMM's current set of discriminants. It was specified that the **customer** user-type would have to access the interface through an ATM, and therefore would not have be able to use a mouse. Interactors that rely on mouse manipulation were never chosen if the customer would have to utilize them. TIMM had no way to account for this information with its current set of discriminants. Nonetheless, the algorithm was able to exploit accidental regularities in the discriminants that it did have access to in order to account for this feature. This is overfitting, and it would probably diminish the algorithm's performance on future design projects. Solutions to the problem of overfitting will be discussed in a later section.

Table 1: Adaptation performance on two experiments

	Experiment 1	Experiment 2
Number of interactors	22	31
Total error correction	98.8%	100%
Transferable error correction (The percentage reduction of the initial number of errors for the opposite example when using an initial decision tree trained on this example)	16%	36%
Preventative error correction with adjustment for overfitting	23%	36%

Experiment 2

The second experiment is based on a portion of a real-world user-interface designed for a medical application for one of our clients. It is slightly larger, with 31 interactors. The automatic suggestions made by TIMM were modified as before, but the modifications were intended to reflect the interface as it was actually designed.

Our first concern was to determine whether the adaptation that took place in the banking example would be of use in this experiment. In fact, even with the problem of overfitting, the refined decision tree performed 16% better

than the original hand-crafted decision tree. When some of the overfitting is corrected, as described below, the decision tree performs 23% better than the original hand-crafted decision tree. Regardless of whether the system was "primed" with a refined decision tree, the adaptive algorithm was able to correct 100% of the errors. 77% of the error was due to an incomplete decision tree, and 23% was due to designer preferences.

This example shows that refining the decision tree will yield benefits for the user interface designer. When the banking application was provided with an initial decision tree trained on the data from the medical application, the gains were even more marked: the refined decision tree performed 36% better than the hand-crafted tree, and there was no overfitting. The banking example was deliberately designed in such a way as to bring about overfitting, but in the real-world example, overfitting was not a problem.

ALGORITHM ISSUES

Complexity

The adaptive algorithm as described searches the entire space of possible changes for the alteration to the decision tree that would be most advantageous. One might question whether such an exhaustive-search algorithm imposes too high of a cost in terms of search time. In fact, the time complexity of the algorithm only grows with the square of the total number of cases. Given the present rate of growth of available computing power, we do not foresee any problems with the complexity of the algorithm.

Local Minima

The adaptive algorithm performs a greedy hill-climbing search for the most immediately rewarding modification to the decision tree. In other words, the algorithm does not look for *sequences* of operations that might be beneficial; rather, it looks ahead only one step, and then applies the single operation that is most beneficial in the short term. In theory, this could lead to problems of local minima. In this case, a local minima would mean that the adaptive algorithm had arrived at a state where no *single* adaptive move would permit further improvement, but where a *series* of adaptive moves might find the globally optimal solution. Then the adaptive algorithm would fail to make the correct modifications to the decision tree, leaving it in an imperfect state. We acknowledge that this is a possible danger. But in practice, local minima have never appeared in any of our tests. The algorithm was able to correct 98.8% of the error in the first example in this paper and 100% of the error in the second example.

Overfitting

The most serious problem is the danger of overfitting to the training data. As mentioned above, this occurred in one of our test cases, resulting in somewhat diminished generality of the adaptive changes. We propose three possible solutions for this problem.

1. **More discriminants**. The more information we can take into account, the less likely overfitting will become. Further testing should point us towards additional discriminants to be taken into account. However, it should be noted that adding discriminants can actually lead to *more* overfitting, if they are not well-chosen. If a discriminant bears no relation to the problem at hand, then it can create the kind of accidental regularity that is often responsible for overfitting. Moreover, additional discriminants increase the size of the tree, making it harder for the user to browse and increasing the runtime of the adaptive algorithm. Therefore, new discriminants should not be added haphazardly; only discriminants that are likely to be useful much of the time should be added.

2. **A sensitivity threshold**. This is to prevent the adaptive algorithm from making any changes unless a benefit greater than some threshold can be achieved. The idea is that the adaptations that result in overfitting are likely responses to one-time idiosyncrasies, whereas adaptations that correct several errors at once indicate general features that are likely to be applicable in the future. As mentioned above, our first attempts to apply this approach met with some success. The generality of the changes increased from 16% to 23%. There is a cost involved: the adaptive algorithm no longer achieves anything near the 98.8% success it had on the training case. Performance on the training example declines to 51%. The sensitivity threshold causes the algorithm to pass over some useful adaptations as well as the dangerous ones that the threshold was meant to filter out. However, if the cases that lead to these useful adaptations recur in later examples, eventually the threshold will be overcome and the necessary adaptations will be made.

3. **User advice**. Another approach would be to simply ask the interface designer for input as to whether and how the tree ought to be changed. The algorithm would then describe the cases involved and ask for advice. The interface designer could recommend paying attention to a particular discriminant, or the designer could simply advise ignoring these cases altogether.

RELATED WORK

Several systems have attempted to automatically generate user interfaces in a model-based environment. UIDE [1] is one of the first model-based user interface design systems, and it automatically selects interactors on the basis of data type—one of the discriminants we consider here. MECANO [5] generates form-based interfaces automatically, and performs interactor selection by

considering several discriminants, including type, cardinality, and the number of allowed values. TRIDENT [12] introduces a comprehensive decision tree that takes into account a broad set of discriminants. These systems represent progress towards automated user interface design, but they do not incorporate adaptation.

Quinlan has developed ID3 [10], an algorithm for decision tree induction. ID3 is based on information theory, and it is intended for batch learning problems that start from scratch with a large number of examples. Our approach favors theory-refinement and sensitivity, so ID3 was not an acceptable choice as an induction algorithm. Maclin and Shavlik present a theory-refinement algorithm that utilizes user advice to minimize the amount of training time necessary [4], but their algorithm is for connectionist learning systems, which we have ruled out because of their poor human-comprehensibility. We were unable to locate a sensitive theory-refinement algorithm for decision tree induction in the literature; ours may be one of the first.

Inference Bear [2] is a programming-by-demonstration application that infers design specifics from user-generated snapshots. Inference Bear uses adaptation to generate a custom user interface by observing the behavior of the interface designer. But whereas Inference Bear is designed to infer *application-specific* design knowledge, our approach seeks to induce and refine general principles of user interface design. Every time a designer begins to design an interface with Inference Bear, it starts afresh. In contrast, the adaptive version of MOBI-D is smarter every time it runs.

FUTURE WORK

The problem of overfitting merits further research. Of particularly interest is the prospect of incorporating user advice to improve the applicability of the adaptation algorithm. Human designers learn better when taught, and there is ample reason to believe that machine learning would also benefit from perspicuous advice. We envision a situation where the interface designer can set a flag in the "preferences" indicating how often they wish to give advice: never, sometimes, or always.

We also hope to apply the methodology described in this paper to other aspects of model-based user-interface design: dialog layout and application structure, for example. MOBI-D provides a formal language to describe both of these phases of design and we believe that adaptive automation could be successful in these domains as well. We also believe that the methodology described in the paper could be applied to a variety of design problems outside of user interface design, and we feel that future research along these lines would be productive.

CONCLUSIONS

We believe that adaptation will be a necessary component of any system that attempts to solve real-world design problems. In this paper, we have presented an automated system that incorporates adaptation to perform interactor selection in user-interface design. Our system is distinguished by the four features that make up the acronym **SURF**: **S**ensitivity to a small number of examples, **U**nderstandability to the interface designer, **R**efinement instead of learning from scratch, and **F**ocus on those aspects of design that are particularly amenable to automation. Preliminary experiments indicate that our adaptation algorithm does provide significant gains in performance. We believe that this methodology can be applied successfully to other areas of user interface design in the future.

ACKNOWLEDGEMENTS

We thank the reviewers for their careful reading and helpful comments. We thank Hung-Yut Chen, Eric Cheng, James J. Kim, Kjetil Larsen, David Maulsby, Justin Min, Dat Ngoyen, Tunhow Ou, David Selinger, and Chung-Man Tam for their work on the implementation and use of MOBI-D.

REFERENCES

1. Foley, J., et al., UIDE-An Intelligent User Interface Design Environment, in Intelligent User Interfaces, J. Sullivan and S. Tyler, Editors. 1991, Addison-Wesley. p. 339-384.
2. Frank, M., Sukaviriya, P., and Foley, J. Inference Bear: Designing Iinteractive Interfaces Through Before and After Snapshots, in Proc of the ACM Symposium on Designing Interactive Systems, pages 167-175, (Ann Arbor, Michigan, August 23-25) 1995.
3. Janssen, C., Weisbecker, C., and Ziegler, J. Generating User Interfaces from Data Models and Dialogue Net Application, in Proc. of InterCHI'93. 1993: ACM Press.
4. Maclin, R. and Shavlik, J. Creating Advice-Taking Reinforcement Learners. Machine Learning, 22:251-281, 1996.
5. Puerta, A. R. The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development, in Proc. of CADUI96: Computer-Aided Design of User Interfaces. 1996. Numur, Belgium.
6. Puerta, A. R. A Model-Based Interface Development Environment. IEEE Software, (14) 4, July/August 1997, pp. 40-47.
7. Puerta, A.R., Cheng, E., Ou, T., and Min, J. MOBILE: User-Centered Interface Building. CHI99: ACM Conference on Human Factors in Computing Systems. Pittsburgh, May 1999, in press.
8. Puerta, A.R. and Eisenstein, J. Towards a General Computational Framework for Model-Based Interface Development Systems. IUI99: International Conference on Intelligent User Interfaces, Los Angeles, January 1999, in press.
9. Puerta, A.R., Eriksson, H., Gennari, J.H. and Musen, M.A. Model-Based Automated Generation of User

Interfaces. In Proc. AAAI 94 (Seattle, July 31- August 1, 1994), AAAI Press, pp. 471-477.

10. Quinlan, J.R. Induction of Decision Trees. Machine Learning, 1: 81-106, 1986.
11. Tam, R.C., Maulsby, D., and Puerta, A.R. U-TEL: A Tool for Eliciting User Task Models from Domain Experts. IUI98: International Conference on Intelligent User Interfaces, San Francisco, January 1998, pp. 77-80.
12. Vanderdonckt, J. M. and Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection, in Proc. of InterCHI'93. 1993: ACM Press.
13. Vanderdonckt, J. and Bodart, F. The "Corpus Ergonomicus":A Comprehensive and Unique Source for Human-Machine Interface Guidelines, in "Advances in Applied Ergonomics", Proceedings of 1st International Conference on Applied Ergonomics ICAE'96 (Istanbul, 21-24 May 1996), A.F. Ozok & G. Salvendy (Eds.), USA Publishing, Istanbul - West Lafayette, 1996, pp. 162-169.
14. Vanderdonckt, J. Advice-Giving Systems for Selecting Interaction Objects, in Proc. Of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS ’99 (Edinburgh, 5-6 September 1999), N. Paton (ed.), IEEE Computer Society Press, Los Alamitos, 1999, to appear.