AI Methods for Automatically Generating User Interfaces

Stephen Ramage* and Kalan MacRow+

Department of Computer Science
University of British Columbia
*sjr@sjrx.net, +kalanwm@cs.ubc.ca

Author's note: The works of Krzysztof Gajos and colleagues at the University of Washington feature prominently herein and, we feel, proportionately to their significant contributions to this field of research.

Abstract

The automatic design, layout and generation of user interfaces for computer applications is a diverse research area that spans information science, human computer interaction, artificial intelligence and graphic design. To many, even today, user interface design is best left to the intuition of experienced creatives. However, the idea of approaching UI layout and generation more systematically and quantitatively was explored at least as early as 1990 (Zanden and Myers). This paper presents a critical overview of the core ideas and techniques that have contributed to modeling user tasks in applications, automatic metric-based evaluation of user interfaces, and solving the problem posed by selecting an optimal interface from an exponentially large array of possible layouts. We present a review of the motivation, implementation and evaluation of several key papers, as well as a commentary on the future directions of this increasingly relevant interdisciplinary field of research.

Introduction

The GUI designer attempts to balance a number of complex and competing objectives including aesthetics, simplicity, ease of use, functionality and accessibility. It is an esoteric job, and we observe that historically the aesthetic component of this process has been of enough import to warrant granting the UI designer creative license in the form of immunity from the standard rigours of engineering. Some, however, have proposed techniques that bring to bear the quantitative tools of artificial intelligence and the contributions of human factors on the problem of user interface design and construction. And indeed, although much remains to be done, incremental but promising progress in the pursuit of automatically

generating user interfaces implies that the widget-wielding designer/alchemist of today will play a much less central role in the future of human-computer interfaces. We present a critical review of methods and results described by a canonical set of papers focused on decision-theoretic UI optimization, evaluation and generation.

§1. Background

Much of the work reviewed here applies general concepts, tools and models from artificial intelligence, human factors, human computer interaction, information science, and planning. Many of the specifics are beyond the scope of the papers we discuss, and are certainly beyond the scope of this review. However, in this section we touch on few notable ideas that will be critical to understanding subsequent sections.

1.1 Fitts' Law

Fitts' law aims to model the time it takes for a user to move their mouse pointer from rest to a target object. It states that the time required is proportional to the logarithm of the quotient of the width of the target and the distance to the target. It is arguable whether Fitts' law applies to everyone, in particular Gajos et al. have found it a poor model of individuals with motor-impairments (Gajos et al., 2007).

1.1 Widgets and Layouts

Many of the papers discussed concern themselves with the composition of user interfaces in terms of widgets and layouts. For the purposes of this discussion, widgets are GUI elements: buttons, drop-down lists, sliders, spinners, list-boxes, etc. Layouts are simply compound elements designed to contain widgets.

1.2 Constraint Satisfaction

Constraint satisfaction is a general process by which a solution to a set of constraints (relationships among some set of variables) is found. A solution consists of a vector of values which when assigned to the variables satisfies the constraints.

1.4 Branch and Bound Search

Branch and bound (B&B) is a search technique for finding optimal solutions in large discrete search spaces. Computing upper or lower bounds on the value being optimized allows pruning large suboptimal regions of the space. B&B has been the general search technique of choice in work that models automatic UI generation as an optimization problem.

1.5 Linear Regression

Linear regression is a general method for modelling data using linear predictor functions. Model parameters are estimated based on a set of observed data points. The derived predictor function can then be used to predict new values of the dependent variable (e.g y) based on previously unseen values of the independent variable, x.

1.6 Support Vector Machines

Support vector machines are supervised learning models used in machine learning for classification, pattern recognition, and regression analysis. An SVM maximizes the width of a "gap" between two categories of the data. The model can then be used to predict which category future data points in the space should be classified as. Mathematically speaking, a *hyperplane* is constructed such that its distance from any data point is maximized. Future data is then classified based on which side of the plane it falls (Cortes et al., 1995).

§2. Layout Appropriateness

Layout appropriateness (LA) is a metric developed for the quantitative evaluation of user interfaces (Sears, 1993). Other work in this area (Jeffries et al., 1991) preceded layout appropriateness but has not enjoyed the same degree adoption. LA attempts to provide a metric that is useful with or without deep task analysis (Anderson, 1990) of users' patterns. The author identifies two broad categories for tools and metrics designed to evaluate user interfaces: those that require task analysis, and those that don't, the so called task-independent approaches.

Task analysis is a deep study of the paths users take to accomplish tasks and the frequency with which they take the various paths possible to accomplish different operations (Anderson, 1990). Unfortunately performing a task analysis, which has it's roots in facilities planning, requires a considerable effort. In the context of user interfaces, operations must be distilled from application features, all possible paths to those features must be identified, and users must be carefully observed to record how often each operation and each path to each operation is used. Sears notes that such an analysis may be reasonable if the project is to improve an existing UI, but for new applications and interfaces the overhead renders it impractical for most designers. In light of this, LA aims for a compromise: include task data if it is available, but work with much simpler-to-obtain task descriptions if it is not. Thus the layout appropriateness metric gracefully degrades depending on the amount of user trace information available. Conversely, quality improves as data becomes available, as it may be on subsequent design iterations.

Sears explains LA as requiring a description of the sequences of widget-level actions users perform and how frequently each sequence is used. LA is computed as the sum of the costs of each sequence weighted

by how frequently the sequence is used. In the original work, the cost is a simple distance function (Fitts' Index of Difficulty) based on how far the user has to move the mouse. Subsequent work (Gajos et al. 2003, 2005, 2007, 2008) has proposed new and different cost functions to better model users with varying abilities. Fitts' law has been a popular cost model in HCI, but has been found inadequate for modeling users with motor impairments or in environments that present challenges to pointing (e.g. buses). With LA, the task description can be based on developer intuitions, a simplified task analysis or, in the best case, a full task analysis. LA has been used to assist designers (Sears, 1995), objectively compare UI designs, and even help automatically generate interfaces (Gajos et al., 2007). Developed over twenty years ago, the methods presented by Sears have been a staple of the research programs of Gajos and others.

2.1 Design and implementation

The implementation of layout appropriateness is reasonably straightforward, but presents the designer with a number of constraints. Among them are that widgets must be of equal size and layout is carried out within a fixed-size grid of cells. Formal constraints can be placed on widgets by the designer to ensure, for example, that "OK" and "Cancel" buttons appear next each other. There are three types of widget constraints available: a widget can be constrained to appear to the right of another widget, below another widget, or in a fixed, absolute position (eg. the top right corner). The algorithm generates all possible layouts of widgets within the grid, and then searches for the layout of optimal cost based on the designer-provided transition diagram. Figure 1 is an example of a simple transition diagram for the Microsoft Write program circa 1993. Nodes are objects in the UI, edges are labelled with the frequency that users move from one object to the other.

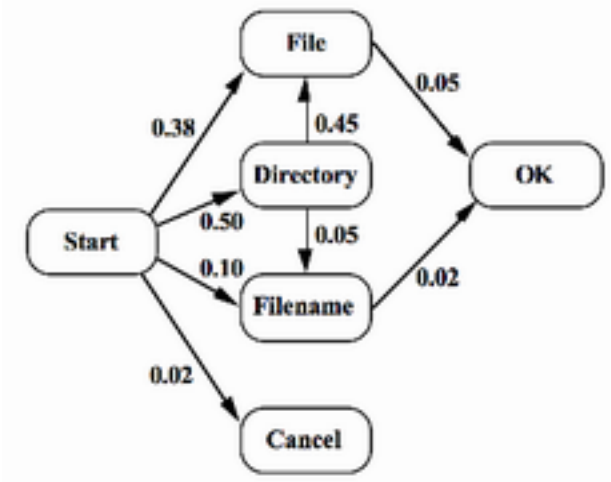

Fig. 1 *Sample transition diagram for MS Write (Sears, 1993).*

Widget constraints are checked as layout cells are assigned concrete widgets during search. Partial solutions that break any constraint can be quickly discarded, or pruned. A branch and bound search algorithm is used so that large branches of fruitless layouts that cannot be optimal are also pruned.

2.2 Evaluation and results

Layout appropriateness was evaluated by creating several layouts of one (in our opinion moderately complex, considering the period) user interface, each with different LA scores, and then recording human participants' a) time to complete a series of tasks, and b) preferences for the UIs they used. This process was repeated for two different interfaces. In the first case, users performed the best and preferred the LA-optimal layout. In the second case, users prefered the second best layout (second in terms of LA score, that is) but performed the best with the LA-optimal layout. All things considered, the evaluation showed that LA has potential to significantly improve UI layouts from a performance perspective. Though, not surprisingly given the absence of any explicit consideration in the model, user preferences are not well accounted for. However, despite this shortcoming, layout appropriateness is a heavily cited work in the automated UI layout space and subsequent work has proven it to be a useful approach to the layout optimization problem.

§3. SUPPLE: Automatically generating user interfaces

The authors of SUPPLE (Gajos et al., 2005) make the astute observation that as computing becomes increasingly ubiquitous, and the number of platforms and devices upon which users expect software applications to run grows, the current methods of designing UI will not scale to meet new demands. In light of this, they present SUPPLE: a system that can generate concrete interfaces for devices with different display sizes and capabilities from an abstract *functional specification* of the interface. SUPPLE frames the problem of generating UI as one of optimization strongly resembling the formulation of Sears' layout appropriateness. SUPPLE takes as input an *interface specification* (Figure 2), a *device model* and a *user model*. The latter consists of device independent user traces, and may be empty; a further simplification of the already simplified task descriptions employed by Sears. The authors exploit the property of their optimization problem that the objective (cost) function is easily replaced, and consequently can be personalized to the needs of different users. They describe a fast algorithm for generating UI tailored to a specific user on a specific device and evaluate their results with an informal user study.

3.1 Design and implementation

The authors model devices as the tuple $<W, C_D, M, N>$ where W is the set of widgets the device is capable of rendering, C_D is a set of device specific constraints (for example, screen size) and M and N are functions that determine the suitability of widgets for particular contexts on the device. Users are modelled with user traces, which are made device-independent by mapping "trails" of concrete UI widgets touched by a user during an operation into sequences of elements in the abstract interface specification.

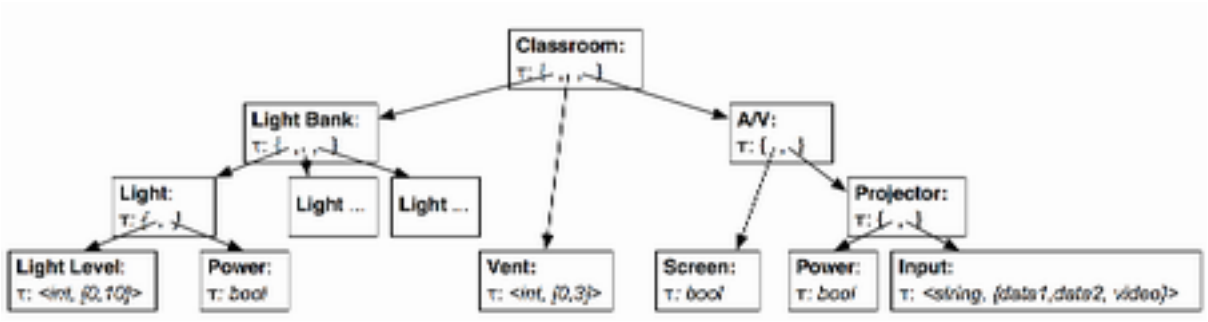

Figure 2. *Functional interface specification of a classroom UI in terms of primitive types. Leaves are widgets and internal nodes are layout containers (Gajos et al., 2005)*

The ultimate goal of the implementation is to render each interface specification element as a concrete widget, optimized to *minimize* user effort in moving through the UI. To this end, a cost function is designed to select the best widget representation for each interface element taking into account several tradeoffs: how good a widget is at manipulating the data represented by the element (the match), the expense of using the widget in terms of movement required (navigational cost), whether or not it is available on the device, and other constraints. A branch and bound search directed by an admissible heuristic (which focuses on the most constrained variable) guarantees SUPPLE will find an optimal concrete interface.

3.2 Evaluation and results

Several optimizations to the search algorithm make SUPPLE's rendering performance quite impressive. In their performance benchmark which had SUPPLE generate the classroom UI described by Figure 2 on three separate touch devices, it found the optimal rendering among 1.8×10^9 possibilities in under 2 seconds. A small, informal user study confirmed that SUPPLE can generate UI equivalent to a human expert when both are given a hypothetical user trace. While the user study component of the evaluation is quite weak (we don't think graduate students in computer science who have taken at least one HCI course constitute experts in UI design) the performance numbers make SUPPLE very usable and promising work. The authors acknowledge that there is yet much to be done, and indeed we review two papers from

Gajos et al. that have built on and extended SUPPLE.

§4. Automatically generating user interfaces adapted to users' motor and vision capabilities

In the first follow-up to their SUPPLE paper, Gajos et al. extend SUPPLE to automatically generate user interfaces for individuals with motor and or vision impairments. The authors argue that at the time of writing (2007) most GUIs were designed with solely the able-bodied user operating in an idealistic environment in mind. Due to a broad range of diseases, disabilities and injuries, many individuals are forced to use specialized assistive devices (eye trackers and head mice for the motor-impaired, physical or software magnifiers for the visually impaired) which are often poorly supported and maintained by such interfaces.

The authors propose SUPPLE++, a system to generate user interfaces tailored to the individual abilities of their user. Like SUPPLE, SUPPLE++ uses a hierarchical functional specification of the UI to generate possible layouts and then search for an optimal solution. This paper makes several important contributions to automated UI generation in general, as well some specifically relevant to generating accessible interfaces. Firstly, they present a set of features for modelling a user's motor abilities, as well an accompanying algorithm for selecting a set of features that best model a particular user. Second, they express the problem of automatically generating GUIs as an optimization problem where the optimal interface will minimize the user's expected movement time based on a model customized for the user. Finally, they present a user study showing that personalized UIs can dramatically improve users' performance and even allow them to accomplish some tasks that they previously could not (Gajos et al., 2007).

4.1 Design and implementation

SUPPLE++ was intended to be deployable in the real world, and as such the authors aimed for it to be easy to setup, configure and maintain. Their goal was to create a system that could generate highly legible UIs capable of rearranging their contents to fit a user's display. The UIs should strike a balance between complexity, functionality, and difficulty to serve users with a diverse combination of motor and visual impairments. For the visually impaired, they hoped to provide a more intelligent solution than simple magnification: dynamic, real-time zoom. Of course these are ambitious desiderata, and although this was preliminary work their results (detailed in the following section) were indeed promising of their methods.

The first stage of configuring SUPPLE++ involved modelling the pointing performance of users. To accomplish this, the authors had eight participants carry out a large array of pointing tasks from the ISO 9241-9 standard and subsequently computed Fitts' law parameters for each. They found that Fitts' law was a reasonable model for the able-bodied (i.e. unimpaired) participants but not a good fit for those

using specialized assistive devices. For example, in the case of participant ET01 who used an eye-tracker, distance only marginally affected the time it took to acquire a target. For HM01, a user who employed a head mouse, distances larger than 650 pixels resulted in a dramatic decline in performance--that is, a dramatically longer amount of time to acquire the target object in the interface. From these results the authors conclude that Fitts' law is inadequate for modelling motor impaired users; it is an empirically poor fit (Gajos et al., 2007).

To replace Fitts' law, the authors identify several features including various transformations of the variables involved in Fitts' law (width of the target, and distance to the target), a constant term, and Fitt's index of difficulty itself. For each user, they find the best set of features to include in the personalized model and then train a regression model that is linear in the selected variables. In SUPPLE, the cost function being minimized was expressed in terms of match quality of widgets as well as their navigational cost. SUPPLE++ introduces a more complex cost function to consider users' expected movement time (EMT), as seen in Figure 3.

$$
\begin{aligned}
\$(R(S_f, s), T) &= EMT(R(S_f, s), T) \\
&= EMT_{nav}(R(S_f, s), T) \\
&\quad + \sum_{n \in S_f} EMT_{manip}(R(n, s), T)
\end{aligned}
$$

Figure 3. *The EMT cost function of SUPPLE++ (Gajos et al. 2007).*

In this new cost function, EMT_{nav} represents the time to navigate the interface, while EMT_{manip} is the expected time to manipulate a widget. The continuous variable s is the minimum target size, however, in practice the authors discretize this parameter to five pixel units making the optimization tractable without stochastic methods.

In order to make the problem amiable to branch and bound search it was necessary that a lower bound on EMT_{nav} be computable. To this end, the authors cleverly propagate a minimum bounding rectangle from the leaves of the functional specification (widgets) up to interior nodes (layouts). In this way they are able to compute a lower bound on the dimensions, and thus the expected navigation time, of any compatible layout instantiation. The ability to employ branch and bound significantly improved search performance, reducing the time to find optimal layouts from days or hours to seconds.

Finally, and almost in passing, they offer an elegant, but not fully realized, solution to accommodate visually impaired users. By allowing them to adjust the visual cue size, SUPPLE++ will dynamically increase or decrease the zoom level of the interface, reflowing the widgets smoothly as needed: a kind

of intelligent magnification not unlike the functionality found in modern web browsers (see A1 for a demonstration video). The authors show an impressive attention to usability with the insight that interfaces rendered in quick succession as the visual cue size is adjusted should not appear drastically different from one another. To ensure this property holds, the cost function in Figure 3 is augmented with a term that penalizes layouts that are significantly different from the current one.

4.2 Evaluation and results

A small user study served as the only evaluation of this preliminary work. Five of the original eight participants were presented with a set of GUIs and asked to perform a series of tasks. One of the GUIs was generated using SUPPLE++ specifically for the user, another was a baseline generated using SUPPLE. A third was also generated by SUPPLE++, but personalized for another user in the group. Their performance and preferences were recorded. The results were promising, but not without noticeable shortcomings. The personalized interfaces increased users' performance by 20% on average. Half of the users were fastest with a personalized UI, and just over half (60%) of the users preferred their personalized UI over all of the others. In a number of cases, however, users performed the best with UIs optimized by SUPPLE++ for another participant. It seems apparent, and the authors note, that a bigger user study capturing a larger range of impairments will be necessary future work. Also, the true cost of manipulating some complex widgets (such as list boxes) was not captured in the model. A follow up paper discussed in section 6 addresses these concerns.

§5. Preference elicitation for interface optimization

Following SUPPLE, Gajos et al. developed ARNAULD, a tool for deriving cost (or utility) functions from interactive preference elicitation sessions with users. Named for the seventeenth century French mathematician who first applied the principle of maximum expected utility, ARNAULD helps designers of decision-theoretic optimization systems avoid the challenging and error prone process of calibrating a factored cost function (Gajos et al., 2005). Acting on the insight that optimization is an increasingly popular technique in the literature, the authors present a fast machine learning algorithm for calculating the weights in a standard formulation of the factored utility function based on preferences elicited from users. They also present two approaches to generating high information gain questions for the process of preference elicitation. SUPPLE (Gajos et al. 2004) is used as a running example throughout the paper to demonstrate how ARNAULD can iteratively build a cost function that allows SUPPLE to generate satisfactory UIs. While ARNAULD itself is not directly concerned with automatic UI generation, we feel that it makes a uniquely important research contribution and is worthy of discussion in this context.

5.1 Design and implementation

The implementation of ARNAULD is multifaceted: it provides two approaches to generating preference elicitation questions as well as a new machine learning algorithm (*maximum margin learner*) that iteratively calculates the parameters of a utility function (Figure 4).

$$\mathbf{S}(o) = \sum_{k=1}^{K} u_k f_k(o)$$

Figure 4. *The standard utility function (Gajos et al., 2005).*

In this form of the utility (or cost) function there are K factors in the system, each represented by f_k and weighted by u_k. The *maximum margin learner* algorithm aims to calculate each u_k subject to user preferences. They base their algorithm on the methods of support vector machines (Cortes et al., 1995). Very fast linear programming techniques allow the algorithm to outperform computing a Bayesian estimate of the weights given users' stated preferences. Covering the precise details are beyond the scope of this review, but the authors note that the Bayesian technique relies on Metropolis sampling, which they demonstrate to be unacceptably slow for an interactive application.

Two question generating techniques (*weight-based* and *outcome-based*) for eliciting high information yield questions are presented. In the context of SUPPLE, *weight-based* generation "enumerates all pairs of different renderings of of any *single* element in the interface specification of an example" (Gajos et al., 2005), the queries are then ranked and the "best" is selected. It is not clear how this ranking is accomplished. *Outcome-based* generation, again in the context of SUPPLE, might involve "taking an element from one functional specification and considering that element in isolation" (Gajos et al., 2005). All such possible queries are enumerated and a scoring function is used to selected the best one. Finally, a UI for actually displaying questions and submitting answers is included.

5.2 Evaluation and results

The evaluation of ARNAULD is reasonably thorough. An informal user study evaluates how easy it is for relatively savvy users to iteratively develop cost functions using ARNAULD. The feedback from developers and sophisticated users was generally quite positive. Also, the maximum margin learner algorithm was benchmarked against the Bayesian approach described earlier. The latter required far more samples to achieve accuracy and took up to 40 seconds, compared to just 200 milliseconds for the authors' machine learning algorithm. The question generation algorithms were evaluated by setting a known target utility function and generating UIs (using SUPPLE) after each question-iteration using the weights calculated to that point. The number of questions asked before the learned function converges with the target is an indicator of the quality of the question generation method. For both weight-based and

outcome-based generation algorithms, it only took 25 iterations before the learned and target functions were “almost identical” (Gajos et al., 2005).

Optimization appears to be a mainstay of automatically generating user interfaces, and given the impressive qualitative and quantitative results of this work we feel that it makes a very practical contribution to the field.

§6. Improving the performance of motor-impaired users with automatically generated, ability-based interfaces

In follow-up work (Gajos et al., 2008), the authors compare improved ability-based user interfaces generated with SUPPLE++ with interfaces created using SUPPLE and preferences elicited using the ARNAULD preference elicitation engine (see Figure 5). The improvement to SUPPLE++ addressed one of the key limitations mentioned in the original SUPPLE++ paper: that the model poorly predicted the cognitive overhead of list-box and other complex widgets. SUPPLE++’s ability elicitation phase was extended to include a section interacting with list widgets, which had never been formally evaluated before. The authors used ARNAULD to display a series of functionally identical widgets to users, and had users rate their preference between them.

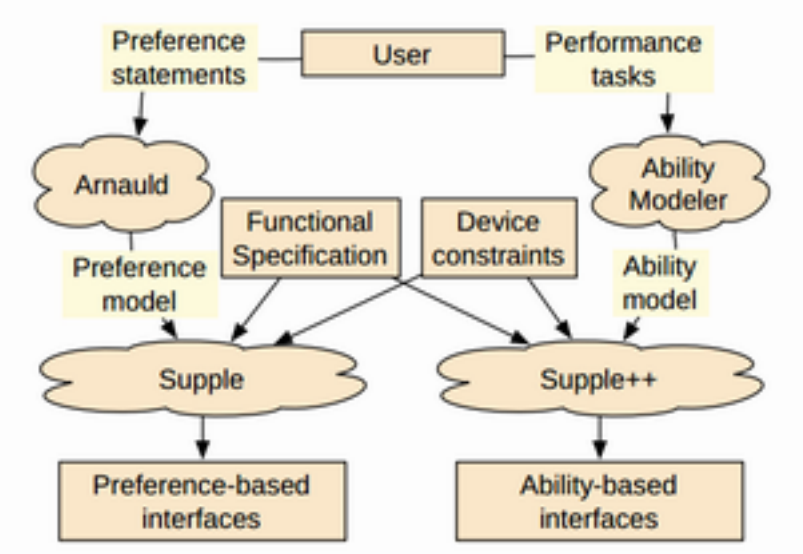

Figure 5. *Preference Based Versus Ability-Based Interfaces (Gajos et al., 2008).*

6.1 Evaluation and results

The authors validate their methods with the following results. Users with motor-impairments experienced a 10% improvement using preference based interfaces, and a 28% improvement over ability based interfaces. Able-bodied users had a 4% improvement with preference based interfaces and an 18%

improvement with ability-based interfaces. Able-bodied users recognized that the ability based interfaces were much easier to use, but did not prefer them because they were “uglier”. Both sets found the ability based interfaces easiest to use, followed by the preference based interfaces, and lastly the baseline interface. Motor-impaired users had no aesthetic preference between interfaces, and consequently viewed the ability based interface as the strongest.

6.2 Strengths and weaknesses

Being a follow-up paper to SUPPLE++, this work seemed very strong and answered many of the previous questions in SUPPLE and SUPPLE++. In terms of weaknesses, the authors mention that their approach with SUPPLE++ is not yet general purpose, and so the results are hardly practical at this point, as no real user interfaces can be readily adapted. One problem with their experimental user study is that they guided participants with visual cues of how to interact with the UI. Because it is guided, it is unclear whether the UI metaphors created by SUPPLE or SUPPLE++ are easy to use for people who have to use a new application and or have to use the same application over a long period of time. While the interaction is doubtless easier, the related problem of uncued user interaction, which we assert to be important, is left open for future work.

§7. GADGET: A toolkit for optimization-based approaches to interface and display generation

The original GADGET paper (Fogarty and Hudson, 2003) was earlier work done in a similar to spirit to that of Gajos et al. with preference elicitation. Optimization-based approaches to generating user interfaces showed potential (Sears, 1993), however, the mathematical complexity of formulating optimization problems might represent a barrier to entry for some developers. As Gajos et al. argued in their preference elicitation paper, constructing utility functions for decision-theoretic optimization problems is challenging and error prone even for confident experts. GADGET presents an “experimental toolkit” (Fogarty et al., 2003) for integrating interface optimization functionality into applications. It provides a set of simple, high-level abstractions that programmers can use to achieve sophisticated optimization behaviour in their user interfaces without having to implement many complex algorithms. Optimization is a compute-intensive task that might often be unreasonably slow to run on commodity desktop machines, however, the authors appeal to Moore’s Law suggesting that we can expect optimization and other (presently) infeasible operations to be commonplace in the near future.

7.1 Design and implementation

Designing an optimization program using GADGET requires the programmer to define three components: an *initializer*, *iterations* and *evaluations*. The initializer does essentially what its name implies: it provides

an initial solution to begin optimization from. Iterations do the work of transforming one possible solution into another, which may involve application logic and or random steps. Evaluations are responsible for evaluating the "goodness" of a solution. Each evaluation represents one dimension of goodness, and is assigned a weight by the programmer. The toolkit comes with a selection of reusable library iterations and evaluations. One very attractive feature of GADGET is that instead of forcing developers to express their evaluations and iterations in strict mathematical formulae, arbitrary code can be used. As the authors point out, this allows programmers to embed the well-understood aspects of their algorithm (e.g. widget spacing) in familiar blocks of code, and defer to the GADGET framework for the less well-understood optimization steps. GADGET uses simulated annealing for optimization: a "temperature" variable that starts out hot probabilistically determines whether or not to accept changes that do not appear to represent an improvement (Fogarty et al., 2003). As the temperature variable cools, the system approaches an optimal solution. Operation proceeds by generating an initial solution, and then iterating on it. After each iteration, the newly generated potential solution is evaluated. This solution is then compared to the one it was generated from (the *prior*) using the comparison operation provided by each evaluation. The system favours whichever potential solution scores higher according to the developer-defined evaluations.

7.2 Evaluation and results

The authors step through several example applications in an effort to evaluate the utility of their toolkit. In particular, they focus on the relative benefits of optimization over other types of algorithms (e.g. greedy). While the merits of optimization are clear, the authors seem to downplay the computational cost of using GADGET. Specifically, they consider BubbleMapping which involves arranging related images (e.g tagged with "NYC") into rectangular clusters. Traditionally the BubbleMap layout algorithm has been implemented with a greedy approach, and consequently it sometimes produces jagged borders when the remaining space to fill is not rectangular. Of course re-imagining this as an optimization problem solves the issue of jagged edges, but it is not clear that the overhead introduced by GADGET would be acceptable in practice--while the occasional jagged edge in a photo collage almost certainly *is*. Of course, it is preliminary work and, given the time period, Moore's Law has more than likely come through for them. A second, more convincing example, looks at generating simplified route maps with LineDrive. GADGET is used to break the problem of laying out roads, labels and intersections into several smaller optimization problems, which proves to be conceptually effective. They conclude that optimization in generating user interfaces is an approach that is here to stay. As such, GADGET is a useful tool that will facilitate the adoption of these techniques, and thus their deployment in the mainstream.

§8. Generating remote control interfaces for complex appliances

This paper describes the implementation of the *personal universal controller* (PUC), which is a generic device for interacting with *appliances* and not *applications*. The authors envision interacting with many different types of appliances (e.g. a telephone or a stereo), from a single device, for instance a PDA (PocketPCs and PalmPilots are used in the paper).

8.1 Design and implementation

The appliances have an *appliance adapter* which contain a detailed description of the UI specification and states, and communicates with a lower level network protocol that supports PUC messages. On the other end, the user's device retrieves this specification and uses it to automatically generate a user interface for interacting with the appliance (Figure 6).

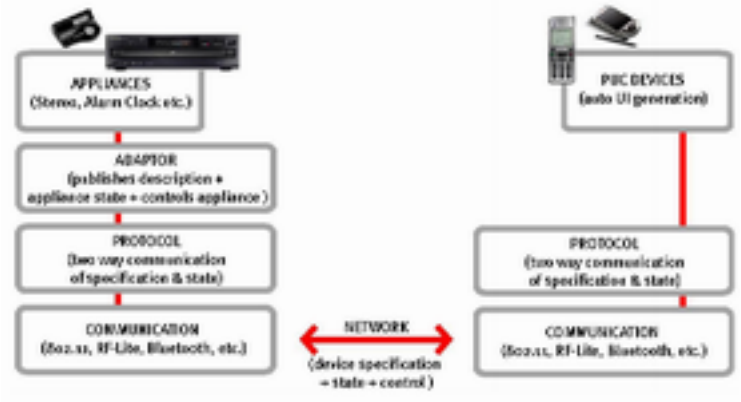

Figure 6. *Architecture of PUC System (Nichols, J., B. et al. 2002).*

The approach taken by the authors in the PUC paper is notably different than those used in SUPPLE in that the interface generation is guided largely through the heavy use of heuristics in converting the specification into a user interface. The initial specification of the user interfaces places the abstract elements into a *group tree*, a tree where leaf nodes with the same parent represent elements that are logically related. Other design heuristics exist to make more suitable user interfaces, such as the following:

> *If a group is found to contain two components that both need their own labels, then a row will be created that has two columns.*

Through the use of these heuristics, the authors demonstrate a very simple algorithm for generating the interface that involves traversing the entire group tree twice: the first traversal computes the size and location of each object in the group tree, and the second actually places the elements. The decision of

which type of UI structure to allocate to each of the abstract elements in the specification is made by a decision tree, which is statically crafted. Their work is heavily influenced by DON (Kim, W. C., and Foley, J. D, 1993), and uses many of the same techniques: static decision trees and simple recursive construction. An example decision for the relative positioning of two widgets on screen is given by Figure 7.

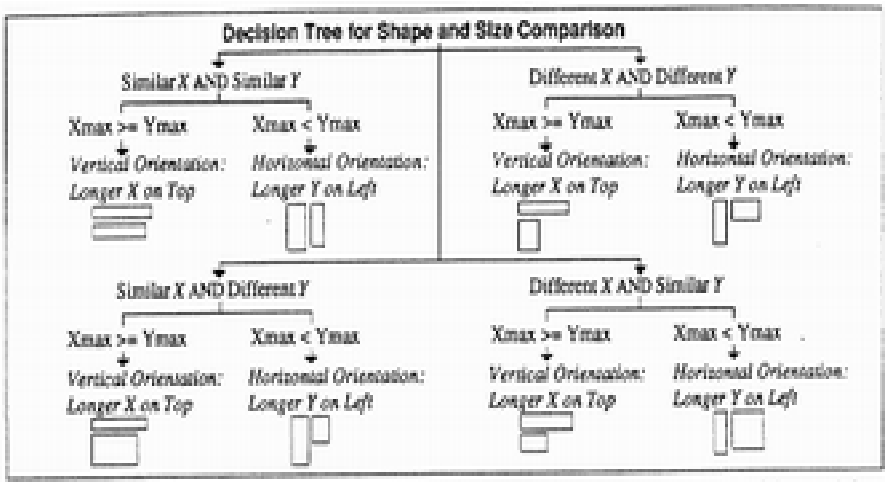

Figure 7. *Decision Tree in DON (Kim, W. C., and Foley, J. D., 1993)*

8.2 Evaluation and results

In developing PUC the authors evaluated the usability of hand-crafted remote interfaces for appliances, and found that they were heavily preferred to the native interfaces of the individual appliances. The authors suggest that this outcome is not particularly surprising, as the actual interfaces are coarse and generally have poorly understood and documented UI metaphors (e.g. holding a button results in something different from pressing it). The authors did not further evaluate the interfaces built by PUC, but it is not clear that this is important for two reasons: the first is that the primary aim of the user interface generation was to provide a single, familiar user interface, not necessarily a comparison with existing interfaces. Second, the heuristics used to build the interface are fairly simple, and the interaction with PUC-capable devices is also fairly simple. It does not seem unlikely that the interfaces designed by these rules would generally be preferred to the "hodgepodge" of physical interfaces. As such, the authors did not need to build a sophisticated system to beat the status quo.

§9. Adaptable and adaptive user interfaces for disabled users in the AVANTI project

AVANTI (Stephanidis et al., 1998) is a system similar to SUPPLE++ that allows a single interface specification to be given, and interfaces to be tailored to individual user impairments with little or

no manual effort. The stated goal in the paper is to define a user interface that is “design[ed] for all”, including disabled users, and targeting multimedia databases.

9.1 Design and implementation

The architecture of the AVANTI UI is a modified web browser that interacts directly with the database containing information that the user is interested in, and another database storing profiles of user types and rules.

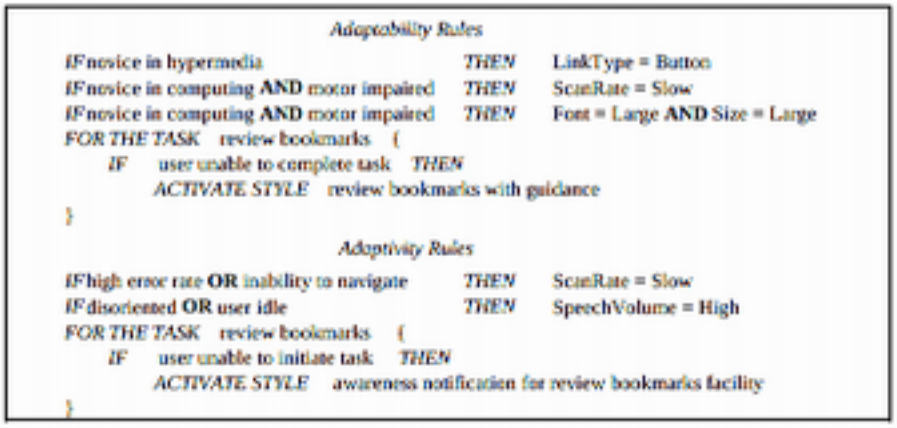

Figure 8. *Rules in the AVANTI knowledge base (Stephanidis, et al. 1998).*

AVANTI uses a knowledge-based representation (Figure 8) of rules in order to adapt its user interfaces for specific users. As the application is used, the knowledge base (KB) is queried based on the profile of the user, and various heuristics to change the style of the user interface or trigger some event in the UI to assist the user. Two sets of KBs are maintained, one with *Adaptability Rules* which are generally applicable to different types of users, and determined based on the state of the application (not the user’s interaction), and the other with *Adaptivity Rules* which are general purpose rules that may be triggered at any time based on some heuristic condition such as the error rate of the user. The result of queries to the KB is generally some predetermined set of changes to the the displayed user interface.

9.2 Strengths and weaknesses

The AVANTI paper is a description of the architecture of the system, and as such does not include any experimental results of the work, nor any conclusions. However, the authors mention an evaluation was underway at the time of writing. The system’s primary weaknesses seem to revolve around the use of a heavily modified web browser that departs from existing systems, making the UI incredibly tailored to a very narrow class of applications. While this web browser can communicate with existing HTTP services, it is suggested in the paper that they would only benefit from limited functionality. Another disadvantage

of this system is that it requires the use of static rules and separate administration of the knowledge base. While the authors do not explicitly state it, one could easily imagine a SUPPLE++ application automatically guiding the user through the process of collecting the information, allowing the user to be entirely self-sufficient. In AVANTI, there is a chicken and egg problem in that a new type of disability cannot be adapted to until the rules for such a disability are established. Consequently, the user interface for inputting the rules could not possibly be adapted to previously unseen needs.

§10. Predictability and accuracy in adaptive user interfaces

One common thread throughout the literature is the suitability of automatically generated user interfaces. Many techniques such as PUC and AVANTI make heavy use of heuristics that guide the UI to a "good enough" solution using predictable and consistent UI metaphors. Others, such as SUPPLE, have many more degrees of freedom. User studies have often resulted in users preferring interfaces other than the aforementioned "optimal" ones.

While the literature on fully adaptive user interfaces like SUPPLE is rather small, the HCI community has been largely hesitant to endorse fully adaptive methods, or provide only lukewarm endorsements (Findlater and McGrenere, 2004 and McGrenere et al., 2008, 2002). In Gajos et al.'s 2008 work with SUPPLE++, the authors detail a simple user study showing that automatic methods can produce superior results to non-adaptive user interfaces, and compare them against a baseline, intelligent agent that supplies only random assistance.

The study used a split interface, where the classic baseline interface was on the left, and a list of automatically generated short cuts appeared on the right. First and foremost the authors were concerned with the trade-off between predictability (i.e. the ability for users to understand the adaptive changes), and accuracy (i.e. the correctness of the the adaptive action). Users were presented with tasks such as finding a button in Microsoft Word, and could either use the baseline interface or the adaptive interface. Two strategies for populating the adaptive interface were considered: random selection, and MRU (most recently used). They also considered the impact of both accuracy of selection, and predictability of selection independently, finding that only accuracy had a statistically significant effect on performance. The authors suggest that this work is merely "initial evidence", but it does provide some defense of the AI methods discussed here against the criticism of our friends in HCI. There remains a substantial amount of work to do in formally, automatically evaluating the usability of interfaces, though some critical work has been done (Ivory et al., 2001).

§11. Future work

The success and practical utility of many of the techniques discussed here depends largely their adoption by mainstream and commercial UI developers. With the goal of widespread adoption in mind, we suggest that there are two important directions for future work. The first concerns refining and extending the methods proposed by Gajos, Stephanidis and others in projects such as SUPPLE, SUPPLE++ and AVANTI. The focus in this vein will be on exploring ways to improve the time and space efficiency of associated algorithms, making them more amiable to the computational resources available on commodity systems. The second important avenue is developing tools and frameworks to help non-experts leverage these advanced UI generation techniques in their projects. Gajos et al.'s work on preference elicitation and Fogarty et al.'s GADGET make promising contributions to this effort, however, both acknowledge that there is much more work to be done. Those doing work in the area concerned specifically with accessibility for impaired users will continue to refine user models. Most are confident that larger user studies including individuals with a more diverse array of capabilities and specialized devices will yield higher quality interfaces.

Lastly, there is debate between whether automatically adaptive approaches like AVANTI are objectively better than static approaches like SUPPLE++ (McGreene J, et. al, 2002). Future work will determine if this is indeed the case or, more likely, that the strengths and weakness of each make them better suited to subtly different applications.

Conclusions

While many different arguments for the value and usefulness of automatically generating UI are made here, we find Gajos et al. in SUPPLE to be the most compelling in 2012. Namely that as the variety of devices upon which we compute increases, traditional methods of UI design and implementation will not be able to keep up. The matrix of devices, platforms, and OS versions that a modern social networking app must support is already too big for anyone except the likes of Facebook, Twitter, Instagram and Google to support. Introducing the dimension of accessibility for impaired users makes the task exponentially bigger. In light of this, we conclude that AI methods for automatically generating high-quality, potentially personalized UI from abstract definitions will be critical to innovation and inclusive design in the years to come.

References

- Gajos, K., Wobbrock, J. and Weld, D. Improving the Performance of Motor-Impaired Users with Automatically Generated, Ability-Based Interfaces. *CHI'08,* Florence, Italy, 2008.

- Gajos, K., Wobbrock, J. and Weld, D. Automatically Generating User Interfaces Adapted to Users' Motor and Vision Capabilities. *UIST'07,* Newport, RI, USA, 2007.
- Fogarty, J. and S. E. Hudson. GADGET: A Toolkit for Optimization-Based Approaches to Interface and Display Generation. *UIST'03*, Vancouver, BC, Canada, 2003.
- Gajos, K. and D. S. Weld. Supple: automatically generating user interfaces. *Proc. IUI'05*, Seattle, WA, USA, 2005.
- Sears, A. Layout Appropriateness: A metric for evaluating user interface widget layout. *Software Engineering,* 19(7):707-719, 1993
- Findlater, L. and McGrenere, J. A comparison of static, adaptive and adaptable menus. *Proc of CHI'04*, 2004, 89-96.
- Findlater, L. and McGrenere, J. (2008) Impact of Screen Size on Performance, Awareness, and User Satisfaction With Adaptive Graphical User Interfaces. *Proc. CHI'08*, ACM Press. p1247-1256.
- Krzysztof Z. Gajos, Katherine Everitt, Desney S. Tan, Mary Czerwinski, Daniel S. Weld. Predictability and accuracy in adaptive user interfaces. *Proc. CHI'08*, ACM Press. p. 1271–1274.
- Carter, S., A. Hurst, J. Mankoff and J. Li. Dynamically adapting GUIs to diverse input devices. *Proc Assets'06* 63-70, New York, NY, USA, 2006. ACM Press.
- Gajos, K. and D. S. Weld. Preference elicitation for interface optimization. *Proc UIST'05,* Seattle, WA, USA, 2005.
- Gajos, K., A. Wu, and D. S. Weld. Cross-device consistency in automatically generated user interfaces. In *Proceedings of Workshop on Multi-User and Ubiquitous User Interfaces (MU3I'05),* 2005.
- R. I. Anderson. Task Analysis: The Oft Missing Step in the Development of Computer Human Interfaces; Its Desirable Nature, Value, and Role. *Proceedings of INTERACT'90* 1051-1054. 1990.
- Nichols, J., B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol. Generating remote control interfaces for complex appliances. *Proc UIST'02*, Paris, France, 2002.
- Ivory, M. Y. and Hearst, M. A. The State of the Art in Automating Usability Evaluation of User Interfaces. *ACM Computing Surveys (CSUR), 33* (4). 470-516. 2001.
- Kim, W. C., and Foley, J. D. Providing High-Level Control and Expert Assistant in the User Interface Presentation Design. *Proc of CHI'93,* 430-437, 1993.
- Sears, A. AIDE: A Step Toward Metric-Based Interface Development Tools. *Proc UIST'95,* 101-110, 1995.

- H. R. Hartson, A. Siochi and D. Hix. The UAN: User-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems.* 8(3) 269-288.
- R. Jeffries, J. Miller, C. Wharton and K. Uyeda. User interface evaluation in the real world: A comparison of four techniques. *Proc of CHI'91* 119-124, 1991.
- W. Kim, J. Foley. DON: User Interface Presentation Design Assistant. *Proc of UIST'90* 10-20, 1990.
- B. V. Zanden and B. A. Myers. Automatic, Look-and-Feel independent Dialog Creation for Graphical User Interfaces. *Proc of CHI'90* 27-34, 1990.
- Stephanidis, C., Paramythis, A., Sfyrakis, M., Stergiou, A., Maou, N., Leventis, A., Paparoulis, G., and Karagiandidis, C. Adaptable and Adaptive User Interfaces for Disabled Users in AVANTI Project. In: S. Triglia, A. Mullery, M. Campolargo, H. Vanderstraeten, M. Mampaey, eds. Intelligence in Services and Networks: Technology for Ubiquitous Telecom Services. Berlin, Heidelberg, New York: Springer, 1998, 153-166.
- McGrenere et al., 2002. An evaluation of a multiple interface design solution for bloated software. Proceedings of ACM Conference on Human Factors in Computing Systems, ACM Press, New York, NY (2002), pp. 163–170
- Cortes, C. and Vapnik, V. Support-Vector Networks. *Machine Learning*, 20, 1995.

Appendix

1. Real-time demo of SUPPLE(++) generating user interfaces. <http://www.youtube.com/watch?v=B63whNtp4qc>