Three Recommender Approaches to Interface Controls Reduction

Nathan Oostendorp
University of Michigan School of Information
Ann Arbor, MI 48109
oostendo@umich.edu

Paul Resnick
University of Michigan School of Information
Ann Arbor, MI 48109
presnick@umich.edu

ABSTRACT
Interface Controls Reduction is the design task of generating simplified interface controls for setting a larger, more complex set of controls. We explore three different empirical approaches to the task: preset sharing, point clustering, and principal component analysis. All three draw on the experience of lead users to recommend simplified controls for others. A case study where they were applied as part of an iterative development cycle with hundreds of users reveals the advantages and challenges of each approach.

Categories and Subject Descriptors
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Design, Experimentation

Keywords
ICR, Slashdot, clustering, principal component analysis, preset

1. INTRODUCTION
"Keep it Simple Stupid" is an often-repeated maxim for interface design, but one that's hard to follow. The underlying functions of the software may involve many user-settable parameters, implying a need for many controls in the UI. To avoid complex interfaces, one approach is to hide less-important controls in nested submenus, "advanced" dialog windows, tabbed menus, and the like. Another approach, which we calls "*Interface Controls Reduction* (ICR)", is to generate composite controls for setting a larger, more complex set of controls. In practice today, designers rely on intuition to choose which controls to highlight or hide and what reduced composite controls to make.

More empirical approaches are now becoming practical. Often software is deployed in stages where different populations of users are allowed access to products still in development. Actions can be recorded and analyzed to show patterns of behavior for these lead users. Whereas users may be willing to "pay" for beta software by tolerating and reporting bugs and obvious flaws, it is also practical to assume they would be willing to share their behavior in the application in order to build a better product. Moreover, the beta users may also be the most motivated to explore product features. Thus, rather than guessing which controls or combinations of them should be elevated to prominence in the interface, these lead users can be given a more complex interface with all the individual controls. Their action can then guide the development of a simpler interface for other users. Clearly, the lead users are unlike other users in at least one respect, their willingness to learn and experiment with complex controls. Despite that difference, their preferences among interface configurations, after exploration, may be reasonably representative of the preferences other users would have, if they, too, incurred the cost of exploration.

We explore three different empirical approaches to Interface Controls Reduction: preset sharing, clustering, and principal components analysis (PCA). They are all "recommender approaches" in that they rely on the explorations and choices of lead users, who interact with the full set of controls, to recommend combinations of settings to other users, much as recommender systems recommend items to users based on the prior experiences of other users with those items. Two of the approaches generate "point" outputs, complete configurations that users can select. The third, PCA, generates "vector" outputs which can be turned into controls such as sliders and knobs that users can interact with.

We first present a hypothetical example, audio equalizer settings, and a real case study, contextual filter settings for the online discussion system at SlashDot. Then we explain each of the three empirical approaches and illustrate with respect to the hypothetical example and the case study. Each section includes a discussion of the challenges in applying that approach. Since we do not think it will always be possible to overcome these challenges, each section also discusses the types of applications the technique is suited to, those where the challenges can be overcome or it is not essential to do so.

1.1 Related work
Presets are a widely deployed interface element. The import/export feature that enables their sharing is less common, however.

The applications of clustering and PCA described here are novel, but have some interesting antecedents. PCA and the related techniques of Singular Value Decomposition and Factor Analysis have been used in information retrieval [1], and in recommender algorithms [2-4]. Perhaps the closest in spirit to our own work is an analysis of UNIX command-line usage using

RecSys'08, October 23–25, 2008, Lausanne, Switzerland.

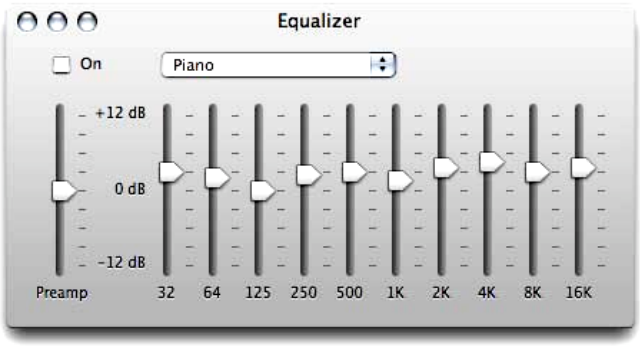

Figure 1. iTunes equalizer with Piano preset applied.

clustering and principal components analysis in order to suggest groupings of commands in documentation or in menus [5]. Unlike our work, however, they did not use the analysis to construct composite commands or controls[1].

Lampe et al [6] analyze the behavior of lead users at Slashdot but did not apply that analysis to generating reduced controls. Other researchers have described interface design techniques that draw on analysis of usage data. Analysis of log data has been used to identify which commands should be mapped to keyboard shortcuts in a text editor [7]. Automated analysis of usage logs from user interface tests has also been used to identify usability problems [8]. Usage traces have also been used to optimize the rendering of controls as interface widgets and to lay out those widgets on different devices [9]; their techniques, however, do not generate new composite controls that simultaneously change multiple underlying controls.

2. Hypothetical Example: Audio Equalizer

Consider a hypothetical task of creating a reduced set of controls for the ten sliders in the audio equalizer found in iTunes, as shown in Figure 1. There is a separate main volume control (not shown). Each of the sliders controls the volume adjustment or offset for an exclusive frequency range. For example, raising the sliders on the left side would boost the bass output while the sliders on the right side would boost the treble output. The iTunes equalizer interface includes presets, one type of simplified control we discuss in this paper. In other audio interfaces, instead of sliders in each frequency range, a variety of reduced controls are sometimes used for equalizer settings, such as bass and treble knobs, a bass boost button, or a "loudness" button that boosts both the lowest and highest frequencies.

3. Case Study: Slashdot Filters

We used empirical data to apply several Interface Controls Reduction techniques to a complex controller for the display of comment threads at the news site Slashdot. Slashdot is an active technology news site. It has a "front page" that shows the most recent stories. When a user clicks on a story, he or she visits a comments page with a threaded discussion. The site gets more than a million page views each day, and the median number of comments per story is about 250[10].

[1] The analogy, though whimsical, would be if they had generated a composite unix command that performed two rm commands, an ls, and ½ of a cat command.

A subset of readers is eligible to assign tags such as "funny", "interesting", "flame", and "overrated". Comments posted by anonymous users start at score 0 while comments posted by logged in users start at score 1 or 2, depending on the author's "karma" in the system. The tags assigned by other readers can move a comment's score up or down, within a range of -1 to 5. The scores are used to sort or filter comments. Figure 2 shows a filtered view of a Slashdot discussion.

3.1 The Initial, Full Controls Interface

We developed a new interface that gives users a lot of control over how comments are displayed, controlling various parameters of a focus+context display [11, 12]. Registered participants in our study automatically interacted with our interface rather than the default interface each time they visited the site Slashdot.org.

The control panel shown in Figure 3 could be opened at any time while reading comments. The "general full" threshold was specified with the top-left set of radio buttons. Comments with scores at or above the full-text threshold are displayed in full. The second set of radio buttons specifies the "general one-liner" threshold. Comments with scores between the one-line and full-text thresholds are displayed in abbreviated form.

The remaining eighteen sets of radio buttons are controls that override the general thresholds. We refer to these collectively as the "extra context" thresholds because they can cause additional comments to be displayed in order to provide context for a reader. The controls all came in pairs; each pair sets a full-text and a 1-line threshold.

The controls in area B are for top-level comments, those that start new threads rather than replying to other comments. In the settings shown, top-level comments need only a score of 4 to be shown in full, while other comments need a score of 5.

In area C, a user could set lower threshold for replies to high scorers or for their parents (the comments the high scorers reply to.) A comment was considered high-scoring if its score was

Figure 2. Filtered Slashdot discussion.

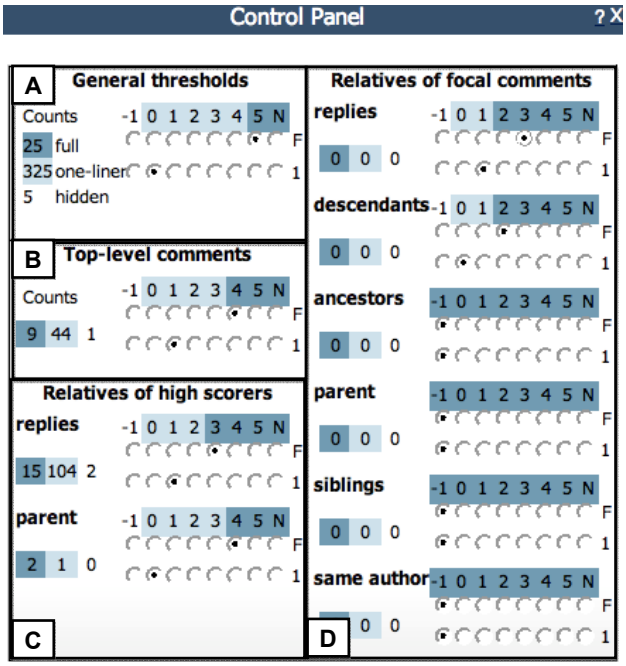

Figure 3. Initial full controls interface for Slashdot comment filtering.

above the general full-text threshold. In area D, a user could also set lower thresholds for display of comments related to a comment that was clicked on.

The motivation for giving users these extra context controls, and analysis of their preferred settings, are not the focus of this paper. Indeed, our conclusion after experimentation was that the additional context controls do not provide sufficient benefit to the browsing process for enough users to be worth including in the production Slashdot interface, even in simplified form. It still provides an illustrative case study, however, of how usage data from early adopters can inform the generation of simplified controls.

Between June 15 and July 13, 2006, 4300 registered users of Slashdot were invited to participate in the experiment. The invitation appeared as a link on the website. Users were recruited in several waves, with slightly different selection criteria. The first group of 500 invitees were users who had posted the most comments on Slashdot recently. Later waves were selected based on having recently logged in, read, or posted comments.

Of these 4300 invited users, 627 completed the informed consent process and the tutorial. Due to bugs in our logging software, the logs for 164 of those users were corrupted, leaving a total of 463 users for whom we have valid data about the settings they chose. Of these, 267 users changed at least one setting.

One participant offered early feedback: "A basic configuration would be nice, with less options that would give simplified basic configurations but still allow an advanced interface that would allow the tweaking available now." It was not obvious, however, what the simplified options should be.

4. Three Recommender Approaches

It is clear that there is an opportunity to learn from the choices of lead users who configure a complete set of controls to suit their needs and preferences. It is not so clear what the concrete mechanisms should be to allow the remaining users to benefit from the explorations of the lead users. We present three approaches.

4.1 Sharing User-created Presets

A preset controls, with a single click, the levels of several individual controls. For example, the iTunes software comes pre-loaded with a number of presets, with names such as "Bass Booster," "Jazz", and "Small Speakers." The "Piano" preset is shown in Figure 1. A user can manually adjust any or all of the sliders and then save that configuration as a preset, giving it a name that is meaningful to the user.

4.1.1 Slashdot case study

We implemented a similar feature for the Slashdot filter settings. A user could set thresholds and then save the current settings as a named preset.

One important design decision was whether every preset should specify all 20 thresholds or whether partial presets could specify values for just some of the thresholds. When a user clicked on a partial preset, the specified thresholds would be changed, but remaining thresholds would stay at their levels before the user applied the preset. While such partial presets would add useful functionality, we were concerned about the complexity of the interface for authoring presets—users would need to understand the concept and then indicate which thresholds to include in the preset. In the end, we opted for the simpler but less expressive option of all presets specifying all 20 thresholds.

The presets feature was introduced more than a year after the initial release and ten months after the release of the simplified "Extra Context" controls described in Section 4.3. 91 users were still using the experimental interface. Between July 13 and Sept. 16, 2007, only nine of them created a preset and only four followed our intended usage pattern, creating multiple presets that they used multiple times. By contrast, during the same period, thirty-six users changed the simplified context controls and ten changed context controls using the full control panel. Thus, we conclude that the named presets were not a very valuable feature for this application.

None of the users chose to export their presets and post them publicly. In most cases, it seems that the presets would not have been useful to other users, even if shared, because the names given to them were not easily interpretable. A few were person specific (e.g., "mine", "default", "Casual", and a person's name). Others were descriptive but conveyed only a general sense of the configuration (e.g., "Minimal", "in-depth", "Uncut"). Only two were easily interpretable, "Threshold 3" and "Threshold 5".

4.1.2 Discussion

There are several advantages of sharing named presets. First it is simple. It is relatively easy to offer the option of saving the current configuration as a preset and to provide an export/import capability to allow sharing. Second, it enables an active sharing process, analogous to the notion of "Active Collaborative Filtering" where users identify specific others to whom them want to recommend an item[13]. Third, any preset that is shared

is truly a preferred setting for at least one user, in contrast to the clustering approach of the next section where the shared preset is an aggregation of many preferred settings and is "close" to all of them but may not actually be preferred by anyone.

There are two main challenges in sharing named presets. The first is user motivation. Unless an individual user wants to switch between different presets, there may be no reason for the user to create and name their configurations. And even if they create presets, they may not have any reason to actively share them with others. The second challenge is the need for recipients to understand what the presets do. The names that preset producers choose may not always convey meaning to the recipients.

Both these drawbacks were apparent in the Slashdot case study. Few users who had already become comfortable with the controls over a long period of time felt the need to create presets, and none were motivated to post their presets for others to examine. And the presets they created did not have names that would have been easily understood had they been shared.

One application where sharing of named presets might be more appropriate is configuring video compression settings. Depending on source material, output size, and platform the user might be asked to choose from various video and audio codecs, resolutions, video and audio bitrates, frame rates, and other processing filters designed to scale down source video for specific applications. A single user might want to prepare video for multiple uses, and thus would be motivated to name their presets. Moreover, many of the names they might choose would describe these situations in a way that would be meaningful to others. For example, for the open source video encoder HandBrake, the community maintains a wiki with presets that have been developed to help users with the complicated controls. Mostly, these are named for the playback devices that they are optimized for [14].

4.2 Point Clustering to Automatically Generate Presets

A second approach, point clustering, aggregates the choices made by a set of lead users to automatically infer a small set of presets to share with other users. It begins with a collection of interface configurations that were helpful to lead users. If a lead user has saved a preset, it is natural to include that preset in the collection. However, even when users have not explicitly saved presets, it is reasonable to include any configuration of all the interface elements that a user has used repeatedly. The intuition behind clustering is to divide the collection of "good configurations" into clusters, with each cluster containing similar configurations. Each cluster will then generate one preset.

In the iTunes equalizer, it is natural to express each configuration as a vector of 10 values, each expressing the level of one of the 10 sliders. The distance between two configurations can then be computed as the L1 or city-block metric, which is the sum of the absolute differences in levels on each of the sliders. Alternatively, L2 or Euclidean distance can be computed as the square root of the sum of the squared differences in levels on each of the sliders.

Given a metric that defines how far apart any pair of configurations are, an algorithm can automatically divide the collection of good configurations into clusters. For each cluster, a centroid or central configuration can be chosen—the centroid is the point that minimizes the sum of distances from all configurations in the cluster. Each centroid can then be provided as a preset to other users. If most of the good configurations are close to one of the centroids, then the presets generated in this way would have provided a reasonably good match for the preferences of most of the lead users.

Of course, if the algorithm is permitted to generate more clusters, and thus more centroids, more of the good configurations will be closer to one of the generated presets. The tradeoff is that the "reduced controls" interface gets harder to understand and use the more presets are offered.

In our example, if iTunes were to automatically collect users' presets and the unnamed configurations that they use frequently it might be possible to cluster them to infer a more useful set of presets to deliver pre-loaded with the software.

4.2.1 Slashdot case study

We applied the clustering approach to the filter thresholds selected by users of the Slashdot control panel in Figure 3. For simplicity, we selected each user's most recent threshold settings as the good configurations. Only users who had changed at least one of the controls were included, since we did not want to infer that a user liked his or her latest setting unless he or she had tried other possibilities.

Before applying clustering, we had to make several decisions particular to this application. First, because of the history of the Slashdot interface, we decided that the simplified interface that we created should include general full-text and one-line controls, plus one or more presets for the remaining 18 context thresholds. This is in contrast to our user-created presets, which controlled the general thresholds as well as the context thresholds.

An inspection of the users' final configurations indicated that many of the context thresholds were set 1 or 2 lower than the corresponding general thresholds. Accordingly, we decided to make the automatically derived simplified controls specify offsets from the general thresholds rather than absolute thresholds on their own. That way, the user could set the context controls once, as offsets, and have the actual thresholds adjust automatically when the user changed the general thresholds.

Thus, we re-expressed each user's final settings for the 18 extra context settings in terms of offsets. To make a consistent scale, offsets were computed as a percentage of the potential offset. For example, if the general threshold was 4, a context control set at 4 or above has a 0% offset, a context control set at -1 has 100% offset and a context control set at 2 has an offset of 40% since it is 40% of the way from 4 to -1. Full-text thresholds were expressed as percent offsets from the general full-text threshold. One-line thresholds were expressed as percent offsets from the minimum of the general and one-line thresholds.

Configurations where either the general full-text or 1-line threshold was set to -1 (the lowest possible value) were excluded because in these configurations it was not possible for the context controls to have lower values than the corresponding general thresholds. 224 users' final configurations met these criteria. This produced a users by controls matrix with 224 rows and 18 columns, with cells indicating the percentage offset for the particular user on the particular controls.

Table 1: Final Percentage offsets for Point Clusters and PCA Settings

	Point Clusters						PCA settings for "extra context"			
Dimension	**None**	**C_1**	**C_2**	**C_3**	**C_4**		**Little**	**Some**	**Lot**	**Max**
Top Level F	3	7	10	25	33		9	15	18	24
Top Level 1L	4	16	8	14	67		10	16	19	24
High Scorer Child F	4	12	6	16	89		12	20	25	33
High Scorer Child 1L	2	29	11	3	83		14	27	35	49
High Scorer Parent F	5	11	21	9	100		17	32	41	55
High Scorer Parent 1L	9	42	26	2	100		24	43	55	74
Clicked Children F	1	20	69	39	0		25	56	75	100
Clicked Children 1L	0	52	100	31	0		35	80	100	100
Clicked Descendants F	0	10	50	18	0		17	39	53	75
Clicked Descendants 1L	0	35	92	13	0		27	63	85	100
Clicked Parent F	0	29	58	20	0		21	49	66	93
Clicked Parent 1L	1	67	100	13	0		34	78	100	100
Clicked Ancestors F	0	7	49	13	0		14	32	44	63
Clicked Ancestors 1L	0	25	100	3	0		21	52	70	100
Clicked Same Author F	0	23	38	22	0		20	45	60	85
Clicked Same Author 1L	0	43	83	13	0		29	68	91	100
Clicked Sibling F	0	4	50	8	0		11	27	37	53
Clicked Sibling 1L	0	20	92	3	0		20	48	65	92

Beginning with the matrix of percentage-offsets, we executed a k-means clustering algorithm using the L2 or Euclidean distance metric. Somewhat arbitrarily, we decided to produce five clusters, with the intention of creating one button or menu item for each cluster's centroid in a simplified interface.

The largest cluster included 176 of the original 224 configurations, those that involved no extra context or only a little. The largest offset on any of the thresholds for this cluster centroid was 9%. Table 1 shows the percentage offsets for five clusters, labeled "none" and C1-C4, for each of the context controls. Note that there is no single ordering of the clusters in terms of how much "extra context" they include. C1 has a lower offset (11%) than C2 (21%) for the full-text threshold for parents of high scorers (i.e., fewer extra comments would display as full-text rather than 1-line); but it has a higher offset for the 1-line threshold (i.e., more extra comments would display as 1-line rather than being hidden completely.) Although C2 gives larger offsets for thresholds in area D of the control panel in figure 2 than to other areas, and C4 larger offsets for controls in areas B and C, we were not able to give meaningful names to clusters other than the first one, for which "no extra context" is a reasonable approximation.

4.2.2 Discussion

The main advantage of point clustering, relative to preset sharing, is its ability to aggregate the preferences of many users, including preferences that are apparent in the configurations users employ but do not name as personal presets. The main advantage relative to the PCA technique, which will be discussed next, is that the cluster centroids are not constrained to be ordered monotonically on any dimension—it is possible to generate a cluster centroid like the Piano equalizer configuration in Figure 1 that is high on some dimensions and another cluster that is high on a subset of those but low on a different subset.

There are two challenges in applying point clustering. The first is to define a metric on configurations such that a user who is satisfied with one configuration will also be pretty satisfied with "nearby" configurations. This may not always be possible. The second is a stronger version of the challenge with sharing named presets—it is difficult to come up with names for the cluster centroids from which shared presets are generated. The latter challenge was most apparent in the Slashdot case study, where we were unable to find meaningful names for the cluster centroids.

The equalizer application seems like one where point clustering might be most appropriate, because the two challenges are less problematic for this application. Configurations with nearly but not quite identical equalizer settings will sound similar to the human ear. And it may not be necessary to name the presets at all. Even in its current incarnation, the names for many of the presets are not that informative, but they may not need to be. A user can try each of the presets while listening to a song and decide quickly what sounds best. By contrast, it takes considerably more effort to evaluate whether a video compression preset is appropriate or a Slashdot thresholds preset, and so the information conveyed in the names is far more important.

4.3 Principal Component Analysis

A third approach, PCA [15], also infers a set of reduced controls from the collection of "good configurations" of lead users. Instead of identifying points in the configuration space (the centroids of clusters) that cover most of the good configurations, it finds a few vectors that can capture most of the variability in that collection. In the case of the iTunes equalizer, it would generate one or a few sliders, not presets as clustering does.

The output of PCA is a set of composite components that are combinations of the original dimensions that corresponded to the base interface controls. If, for example, in the audio equalizer setting, whenever users set the left-most slider higher, they also tended to set the next two sliders higher as well, one composite component might include a positive weight on those three dimensions, and little or no weight on the others. This composite component would then be turned into an interface element such as a slider, whose values would indicate how much of the composite dimension to include. Thus, raising the slider for the composite "bass" dimension described above would simultaneously boost the volume for the three lowest frequency ranges. The amount that each was boosted could vary, depending on the weight that the composite component gives to each of the original dimensions.

Each original configuration can be projected onto the nearest point that can be expressed as a linear combination of the new composite dimensions. It may not be possible, however, to exactly express all of the original configurations using just the reduced set of composite dimensions. For example, it will be impossible to exactly match the Piano configuration in Figure 1 using a "bass" control that adjusts all three of the lowest frequency bands: the two lowest bands have positive values in the Piano configuration but the third does not.

Consider the distance between an original configuration and the closest point that can be generated using the reduced controls as an error measure for the reduced controls. For any fixed number of reduced controls to be generated, PCA will automatically generate a set of composite dimensions that minimizes the sum of those errors. Including more components from the PCA, and thus more controls in the reduced interface, will reduce the total sum of errors, meaning that the controls can be used to come closer to expressing more of the good configurations. On the other hand, more controls imply more complexity in the user interface. At the extreme, if the number of components selected from the PCA equals the number of original interface controls, the reduced controls will be fully expressive but not any simpler than the original ones.

As with clustering, it may be difficult to name and describe the composite controls resulting from PCA in a way that users can understand. To the greatest extent possible, each original interface dimension will load on just one Varimax rotated principal component. When things work out well, each reduced interface element may control a distinct subset of the original interface elements, and there may be a natural name for that collection. For example, with the equalizer controls, the component that affects the three lowest frequency ranges might be labeled as "bass". There is no guarantee, however, that the principal components will always admit such simple names.

While the reduced controls may be difficult to label, the values on each control are ordered. Thus, if the control has an intuitively understandable label, it will be easy for a user to understand that different settings on that control indicate more or less of the thing being controlled.

Sometimes it is desirable to have a few discrete choices for each reduced control, rather than continuous controls such as sliders. For example, it may be desirable to have a "loudness" button that is either on or off, providing a discrete boost to the lowest and highest frequencies, rather than a continuous "loudness" slider. Using clustering, it is possible to automatically determine an optimal set of discrete choices on each principal component. Each good configuration is projected onto its nearest point that can be expressed in terms of the reduced dimensions. Thus, each good configuration generates a preferred level for each reduced control. The set of preferred levels for a reduced control can then be clustered using the technique of the previous section to generate the best set of discrete options for that control.

4.3.1 Slashdot case study

We applied PCA to the same matrix X of offsets from users' general Slashdot thresholds that we used in the clustering analysis. PCA produced a set of easily interpretable composite dimensions. The largest eigenvalues of the correlation matrix X^TX were 7.38 and 2.55, with three more values between 1 and 2, and thirteen more below 1.

With two components retained, after varimax rotation the two components were naturally interpreted as "extra context before clicks" and "extra context after clicks". Thus, we could have provided two extra context controls, one that simultaneously adjusts all the top-level and high-scorer controls from areas B and C of the original control panel in Figure 3, and one that adjusts the clicked-on comments controls (area D). With three components retained, the after-click control would have been further divided into two controls, one that affects the full-text thresholds in area D and one that affects the 1-click thresholds in area D.

Even two or three controls for extra context, however, seemed like too much complexity for the reduced interface. We opted to include a single control, based on the single eigenvector that captured the most variability in user preferences. It captured 41% of the overall variance in the actual settings users selected. The single vector loads positively on all the individual controls, though somewhat more on some controls than others. In order to create discrete options rather than a continuous control, to match the style of the then-standard Slashdot interface, we employed the technique described above of projecting each row of X onto the single eigenvector, then clustering. We labeled the overall control as "Extra Context" and the five discrete options as "None", "A little", "some", "a lot", and "maximum", as shown in Figure 4.

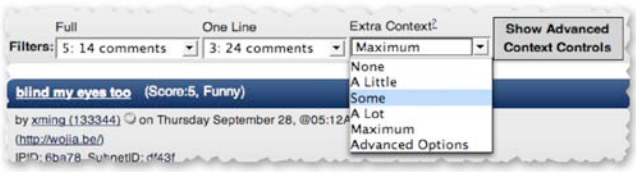

Figure 4. Reduced "Extra context" control

Table 2. Summary of approaches for Interface Controls Reduction.

Name	Preset sharing	Point clustering	PCA
Description	Users save and name complete configurations; export for others to use	Divide users' good configurations into clusters; centroid of each cluster becomes one configuration to share	Apply principal component analysis; each retained component becomes one reduced control
Advantages	· Share with specific people · Exact favorites	· Aggregate configurations · Finds favorites that may not be ordered	· Aggregate configurations · Each control is interpretable as less or more
Challenges	· Motivation to name and share · Personal names may not be publicly understood	· Hard to name the cluster centroids · Requires distance metric matched to user preferences	· Hard to name the dimensions · Requires distance metric matched to user preferences
Suitable Application	· Video compression settings	· Audio equalizer	· Slashdot context filter settings Browser security settings

The right side of Table 1 shows the meaning of each of the labeled options. For example, if the user selected "a lot" of extra context, the 1-liner offset for children of clicked-on messages would be 100% but the 1-liner offset for children of high scorers would be only 35%. Thus, if the 1-line general threshold was 2, the threshold for children of clicked-on messages would be -1 and the threshold for children of high scorers would be 1.

A version of the interface with the simplified "extra context" control was released on August 29, 2006. Most people, who had already configured the advanced settings, did not try the simplified settings during a two week data collection period after the interface was released. Of the 39 users who did try the simplified settings, however, only three had switched back to advanced controls at the end of the data collection and only four had set the "extra context" to none, suggesting that the single control was sufficiently expressive for most users who tried it.

Survey respondents generally praised the simplified controls. One wrote: "I really like the changes in late August, they made it much easier. Especially the extra controls thing, it's so much simpler." Another wrote: "I really enjoy both the advanced and simplified interfaced, great improvement from the first test version!" The interface was not tested with new users, however, so we cannot be sure whether users who had never seen the advanced control panel would have understood the single "extra context" control or liked the alternative settings that it provided.

4.3.2 Discussion

Like point clustering, PCA has the advantage of automatically aggregating the preferences of many users. Relative to point clustering, the main advantage is that the simplified controls are easily interpreted as providing more or less of each reduced dimension.

The main challenge is that it may not be obvious what to name the reduced controls, to convey to users what they are getting more or less of. In addition, like point clustering, dimension reduction depends on the existence of a distance metric among configurations such that nearby configurations are of similar utility to users.

The use of PCA here is analogous its use in personalized recommendations, but with an important twist. For example, the factor analysis approach to recommending items [2] begins with a matrix whose elements are user's ratings of items. That is analogous to our matrix X which gives user's settings (levels) for each of the items (the original controls). Then, the factor analysis approach derives a set of underlying factors (analogous to our components in PCA). Each factor has weights for each of the items, and each person's vector of ratings for the items is approximated by loadings or preferences for each of the factors.

From here, however, the processes diverge. In conventional recommenders, the user never sees any representation of the principal components or factors. It is presumed that users can easily rate some individual items; the purpose of identifying factors or components is to extend a user's expressed preferences to additional items beyond those that the user has rated. If we were to apply the standard recommender system approach directly, we would have users pick their settings on a few of the original controls and infer their preferred settings for the other controls. In our interface controls application, by contrast, each principal component becomes a reduced control that is visible to the user; the user manually inputs his or her preferences using the reduced controls.

Having the user directly interact with the controls derived from the principal components, however, creates a constraint that they must be understandable. PCA worked reasonably well for the Slashdot case because the principal components produced composite controls that mapped to meaningful sets of the original controls. In applications where that is not the case, PCA may not be an effective technique.

Another application where PCA might work well is browser security settings. Modern web browsers include a large number of configurable options about what the browser is allowed to do, ranging from allowing the innocuous inconvenience of popups to the extreme vulnerabilities of running unsigned plugins. Internet Explorer 7, for example, has over 40 configurable parameters, some with multiple possible settings. Many end-users do not understand what all of the underlying parameters

control. Expert lead users might create various configurations but to convey their meanings to end-users it is probably necessary to array them on a single dimension of more or less secure. Indeed, Internet Explorer provides a single reduced control with values Medium, Medium-High, and High; as far as we know, however, the meaning of those settings was not determined through empirical analysis of the settings of expert lead users.

5. Conclusion

Our examples of Interface Controls Reduction show the potential for interface design to move towards a practice based on user-generated data. The three methods we present, user-contributed presets, point-clustering, and PCA, allow for a population of early users to indicate the useful space inside of a complicated interface design space and map it out for subsequent users. Together with other techniques such as automated widget selection and layout [9], our techniques contribute to an emerging toolkit for usage-driven interface design.

Our case study with Slashdot indicates that some advanced users can come to grips with a complicated interface, and generate useful data indicating how to combine large numbers of controls. In this case, we believe PCA was the most useful technique, but in other cases the methods of point clustering and preset sharing would be more appropriate. Table 2 summarizes our assessment of the advantages and challenges of each approach.

Some challenges are evident for ICR to become used widely as a tool for designers. Data acquisition of user behavior is not standardized across applications and toolkits, and must be integrated into an application with substantial development time and effort. Moreover, we had to make a number of application-specific choices in our Slashdot study. As more experience accrues with ICR, analysis and design patterns may emerge that provide more guidance on these application-specific choices. The adoption of ICR could be accelerated by integrating data acquisition into a UI framework, and data analysis into a complementary GUI creation tool. Application interface design as an iterative methodology could then routinely include quantitative user-generated data as a source of feedback.

Fortunately, modern web-based applications using server-side data logging provide an ideal platform for data acquisition from a user population, and these applications can be updated frequently and instantaneously without user interaction. As browser-based GUI applications become more popular, the data needed for ICR could easily become more readily available. This is a promising trend for the adoption of recommender systems for interface design.

6. ACKNOWLEDGMENTS

We thank Rob Malda, Chris Nandor, Tim Vroom, and Scott Collins at Slashdot for courageously allowing us to integrate our experimental interface into the live functioning of the site, and providing technical support for our implementation efforts.

7. REFERENCES

[1] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. and Harshman, R. Indexing by latent semantic indexing. *Journal of the American Society for Information Science*, 41, 6 1990), 391-407.

[2] Canny, J. Collaborative filtering with privacy via factor analysis. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval* (Tampere, Finland, 2002). ACM, 238-245.

[3] Goldberg, K., Roeder, T., Gupta, D. and Perkins, C. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4, 2 (July 2001).

[4] Sarwar, B. M., Karypis, G., Konstan, J. A. and Riedl, J. Application of Dimensionality Reduction in Recommender System- A Case Study. In *Proceedings of the ACM WebKDD 2000 Web Mining for E-Commerce Workshop* (2000).

[5] Hanson, S. J., Kraut, R. E. and Farber, J. M. Interface design and multivariate analysis of UNIX command use. *ACM Transactions on Information Systems*, 2, 1 1984), 42-57.

[6] Lampe, C., Johnston, E. and Resnick, P. Follow the reader: filtering comments on Slashdot. In *Proceedings of the CHI '07* (San Jose, California, USA, 2007). ACM Press, 1253-1262.

[7] Good, M. The use of logging data in the design of new text editor. In *Proceedings of the CHI 85* (San Francisco, CA, US, 1985), 93-97.

[8] Siochi, A. C. and Hix, D. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *Proceedings of the CHI 91* (New Orlean, LA, 1991). ACM Press, 301-305.

[9] Gajos, K. and Weld, D. S. SUPPLE: automatically generating user interfaces. In *Proceedings of the IUI '04 conference on Intelligent user interfaces* (2004). ACM, 93-100.

[10] Lampe, C. and Resnick, P. Slash(dot) and Burn: Distributed Moderation in a Large Online Conversation Space. In *Proceedings of the CHI 04* (2004), 542-550.

[11] Furnas, G. W. Generalized Fisheye Views. In *Proceedings of the CHI '86* (Boston, 1986), 16-23.

[12] Furnas, G. W. A Fisheye Follow-up: Further Reflections on focus + context. In *Proceedings of the CHI 06* (Montreal, Quebec, Canada, 2006). ACM Press, 999-1008.

[13] Maltz, D. and Ehrlich, K. Pointing the way: active collaborative filtering. In *Proceedings of the Proceedings of CHI '95* (Denver, Colorado, United States, 1995). ACM Press/Addison-Wesley Publishing Co.

[14] HandBrake *HandBrake's Built-In Presets, Wiki Page*. City, 2008.

[15] StataCorp *Multivariate Statistics Reference Manual*. Stata Press, College Station, Texas, 2007.