Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition

Björn Hartmann[1], Leith Abdulla[1], Manas Mittal[2], Scott R. Klemmer[1]

[1]Stanford University HCI Group
Computer Science Dept, Stanford, CA 94305
[bjoern|srk|leith.abdulla]@cs.stanford.edu

[2]The MIT Media Laboratory
20 Ames St., Cambridge, MA 02139
manas@media.mit.edu

ABSTRACT

Sensors are becoming increasingly important in interaction design. Authoring a sensor-based interaction comprises three steps: choosing and connecting the appropriate hardware, creating application logic, and specifying the relationship between sensor values and application logic. Recent research has successfully addressed the first two issues. However, linking sensor input data to application logic remains an exercise in patience and trial-and-error testing for most designers. This paper introduces techniques for authoring sensor-based interactions by demonstration. A combination of direct manipulation and pattern recognition techniques enables designers to control how demonstrated examples are generalized to interaction rules. This approach emphasizes design exploration by enabling very rapid iterative demonstrate-edit-review cycles. This paper describes the manifestation of these techniques in a design tool, Exemplar, and presents evaluations through a first-use lab study and a theoretical analysis using the Cognitive Dimensions of Notation framework.

Author Keywords

Sensors, physical computing, design tools, PBD

ACM Classification Keywords

H.5.2. [Information Interfaces]: User Interfaces—*input devices and strategies; interaction styles; prototyping; user-centered design*. D.2.2 [Software Engineering]: Design Tools and Techniques—*User interfaces*.

INTRODUCTION

Sensing technologies are becoming pervasive, and sensor hardware is increasingly diverse and economical. Recent work in physical computing toolkits has lowered the threshold for connecting sensors and actuators to PCs [6, 7, 18-20]; to design self-contained physical interfaces [1, 8]; and to prototype and evaluate the application logic of systems that make use of sensors and actuators [19]. Accessing sensor data from software has come within reach of designers and end users.

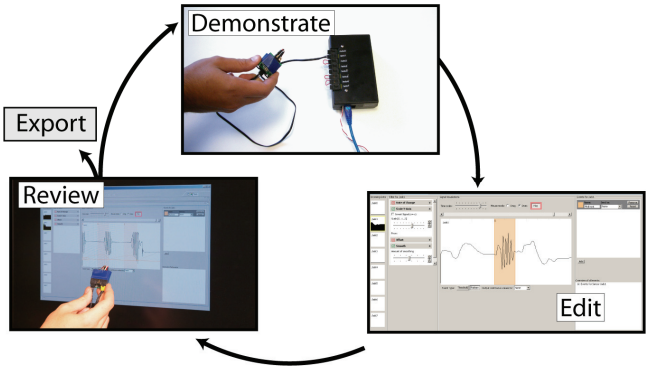

Figure 1. Iterative programming by demonstration for sensor-based interactions: A designer performs an action; annotates its recorded signal in Exemplar; tests the generated behavior, and exports it to a supported authoring tool.

However, our experience of first learning and later teaching physical interaction design, and our experience of deploying d.tools [19] in the classroom, has shown that *specifying the relationship* between sensor input and application logic remains problematic for designers for three reasons. First, most current tools, such as Arduino [1], require using textual programming to author sensor-based behaviors. Representations are most effective when the constraints embedded in the problem are visually manifest in the representation [31]. Thus, numbers alone are a poor choice for making sense of continuous signals as the relationship between performed action and reported values is not visually apparent. Second, existing visual tools (*e.g.*, LabView [2]) were created with the intent of helping engineers and scientists to perform signal analysis; as such, they do not directly support interaction design. This leaves users with a sizeable *gulf of execution,* the gap between their goals and the actions needed to attain those goals with the system [21]. Third, the significant time and cognitive commitments implied by a lack of tools inhibit rapid iterative exploration. Creating interactive systems is not simply the activity of translating a pre-existing specification into code; there is significant value in the epistemic experience of exploring alternatives [22]. One of the contributions of direct manipulation and WYSIWYG design tools for graphical interfaces is that they enable this "thinking through doing"—this paper's

research aims to provide a similarly beneficial experience for sensor-based interaction.

This paper contributes techniques for enabling a wider audience of designers and application programmers to turn raw sensor data into useful events for interaction design through *programming by demonstration*. It introduces a rapid prototyping tool, Exemplar (see Figure 1), which embodies these ideas. The goal of Exemplar is to enable users to focus on design thinking (how the interaction should work) rather than algorithm tinkering (how the sensor signal processing works). Exemplar frames the design of sensor-based interactions as the activity of performing the actions that the sensor should recognize — we suggest this approach yields a considerably smaller gulf of execution than existing systems. With Exemplar, a designer first *demonstrates* a sensor-based interaction to the system (*e.g.*, she shakes an accelerometer). The system graphically displays the resulting sensor signals. She then *edits* that visual representation by marking it up, and *reviews* the result by performing the action again. Through iteration based on real-time feedback, the designer can refine the recognized action and, when satisfied, use the sensing pattern in prototyping or programming applications. The primary contributions of this work are:

- A method of programming by demonstration for sensor-based interactions that emphasizes designer control of the generalization criteria for collected examples.
- Integration of direct manipulation and pattern recognition through a common visual editing metaphor.
- Support for rapid exploration of interaction techniques through the application of the design-test-analyze paradigm [19, 24] on a much shorter timescale as the core operation of a design tool.

The rest of this paper is organized as follows: we first introduce programming by demonstration, and then describe relevant characteristics of sensors and sensor-based interactions to position our work. We provide an overview of the Exemplar research system and describe its interaction techniques and implementation. We then report the results of two evaluations we have employed to measure Exemplar's utility and usability, and discuss related work.

PROGRAMMING BY DEMONSTRATION

Programming by demonstration (PBD) is the process of inferring general program logic from observation of examples of that logic. Because PBD builds program logic "under the hood" and does not require textual programming, it has been employed for *end-user development* [30]. In many PBD systems, the examples or demonstrations are provided as mouse and keyboard actions in a direct manipulation interface (cf. [13, 26, 27]). These systems often take the form of visual programming environments. PBD has been employed in educational software to introduce children to programming concepts; for the specification of macros; and in robotics to specify navigation [5], or grasp motions for assembly arms [29]. Our research builds upon the idea of *using actions performed in physical space* as the example input.

The crucial step in the success or failure of PBD systems is the *generalization* from examples to rules that can be applied to new input [26]. This inference step often leverages machine learning and pattern recognition techniques. But what if the system's generalization does not match the author's intent? Our research addresses the central importance of guided generalization by proposing a *direct-manipulation environment for editing the generalization rules applied to examples performed in physical space*.

SENSOR-BASED INTERACTIONS

This section introduces an analysis of the space for sensor-based interactions from the designer's point of view. Prior work has successfully used design spaces as tools for thinking about task performance [12] and communicative aspects [9] of sensing systems. Here we apply this approach to describe *the interaction designer's experience of working with sensors and authoring for them.* This design space foregrounds three central concerns: the nature of the input signals, the types of transformations applied to continuous input, and techniques for specifying the correspondence between continuous signals and discrete application events.

Binary, Categorical, and Continuous Signals

As prior work points out [33], one principal distinction is whether sensing technologies report continuous or discrete data. Most technologies that directly sample physical phenomena (*e.g.*, temperature, pressure, acceleration, magnetic field) output *continuous* data. For discrete sensors — because of different uses in interaction design—it is helpful to distinguish two sub-types: *binary* inputs such as buttons and switches are often used as general triggers; *categorical* (multi-valued) inputs such as RFID are principally used for *identification*. A similar division can be made for the outputs or actuators employed. *Exemplar* focuses on continuous input in single and multiple dimensions.

Working with Continuous Signals

Sensor input is nearly always transformed for use in an interactive application. Continuous transformation operations fall into three categories: conditioning, calibration, and mapping. *Signal conditioning* is about "tuning the dials" so the signal provides a good representation of the phenomenon of interest, thus maximizing the signal-to-noise ratio. Common steps in conditioning are de-noising a signal and adjusting its range through scaling and offsetting. *Calibration* relates input units to real-world units. In scientific applications, the exact value in real-world units of a measured phenomenon is of importance. However, for many sensor-based interfaces, the units of measurement are not of intrinsic value. *Mapping* refers to a transformation from one parameter range into another. Specifying how sensor values are mapped to application parameters is a creative process, one in which design intent is expressed. Exemplar offers support for both conditioning sensor signals and for mapping their values into binary, discrete, or

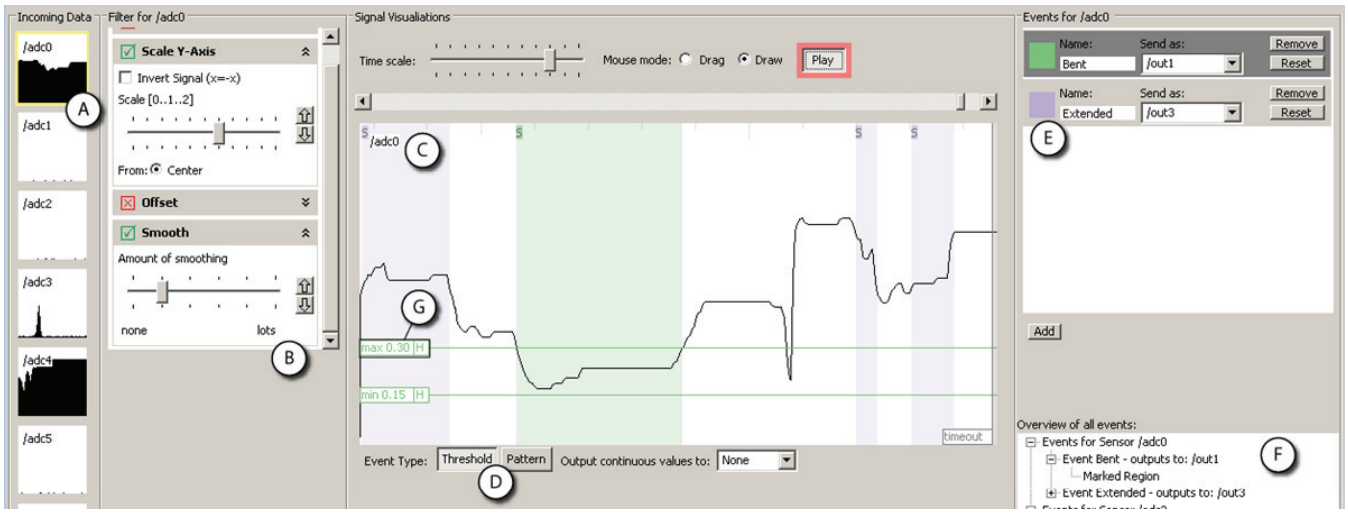

Figure 2. The Exemplar authoring environment offers direct manipulation of live sensor data.

continuous sets. When calibration is needed, experts can use Exemplar's extensible filter model.

Generating Discrete Events

A tool to derive discrete actions from sensor input requires both a detection algorithm and appropriate interaction techniques for controlling that algorithm. The most computationally straightforward approach is thresholding — comparing a single data point to fixed limits. However, without additional signal manipulations, *e.g.*, smoothing and derivatives, thresholds are susceptible to noise and cannot characterize events that depend on change over time. Tasks such as gesture recognition require more complex pattern matching techniques. Exemplar offers both thresholding with filtering as well as pattern matching.

The user interface technique employed to control how the computation happens is equally important. Threshold limits can be effectively visualized and manipulated as horizontal lines overlaid on a signal graph. In our experience, the parameters of more complex algorithms are less well understood. Exemplar thus frames threshold manipulation as the principle technique for authoring discrete events. Exemplar contributes an interaction technique to cast parameterization of the pattern matching algorithm as a threshold operation on matching error. Through this technique, Exemplar creates a consistent user experience of authoring with both thresholding and pattern matching.

In the next section, we describe how the design concerns outlined here—support for continuous input, techniques for transforming that input, and techniques for discretizing input to application events—are manifest in Exemplar's UI.

DESIGNING WITH EXEMPLAR

Designers begin by connecting sensors to a compatible hardware interface, which in turn is connected to a PC running Exemplar (see Figure 2). As sensors are connected, their data streams are shown inside Exemplar. The Exemplar UI is organized according to a horizontal data-flow metaphor: hardware sensor data arrives on the left-hand side of the screen, undergoes user-specified transformations in the middle, and arrives on the right-hand side as discrete or continuous events (see Figure 3). The success of data-flow authoring languages such as Max/MSP attests to the accessibility of this paradigm to non-programmers.

Peripheral Awareness

Live data from all connected sensors is shown in a *small multiples configuration.* Small multiples are side-by-side "graphical depictions of variable information that share context, but not content" [38]. The small multiples configuration gives a one-glance overview of the current state of all sensors and enables visual comparison (see Figure 2A). Whenever a signal is "interesting," its preview window briefly highlights in red to attract attention, then fades back to white. In the current implementation, this occurs when the derivative of a sensor signal exceeds a preset value. Together, small multiple visualization and highlighting afford peripheral awareness of sensor data and a visual means of associating sensors with their signals. This tight integration between physical state and software representation encourages discovery and narrows the *gulf of evaluation,* the difficulty of determining a system's state from its observable output [21]. For example, to find out which output of a multi-axis accelerometer responds to a specific tilt action, a designer can connect all axes, tilt the accelerometer in the desired plane, and look for the highlighted thumbnail to identify the correct channel for her design.

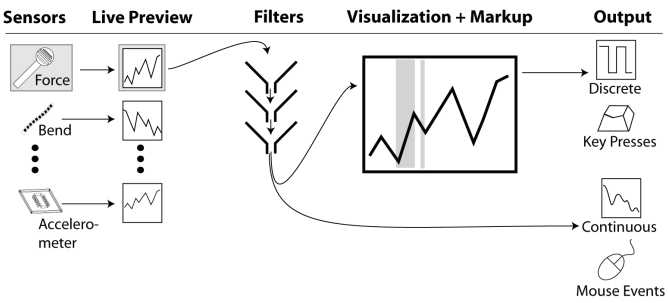

Figure 3. In Exemplar, sensor data flows from left to right.

Constant view of all signals is also helpful in identifying defective cables and connections.

Drilling Down and Filtering

Designers can bring a sensor's data into focus in the large central canvas by selecting its preview thumbnail (see Figure 2C). The thumbnails and the central canvas form an overview plus detail visualization [36]. Designers can bring multiple sensor data streams into focus at once by control-clicking on thumbnails. Between the thumbnail view and the central canvas, Exemplar interposes a *filter stack* (see Figure 2B). Filters transform sensor data interactively: the visualization always reflects the current set of filters and their parameter values. Exemplar maintains an independent filter stack for each input sensor. When multiple filters are active, they are applied in sequence from top to bottom, and these filters can be reordered.

Exemplar's filter stack library comprises four operations for conditioning and mapping — *offset*: adds a constant factor; *y-axis scaling*: multiplies the sensor value by a scalar value, including signal inversion; *smoothing*: convolves the signal with one-dimensional Gaussian kernel to suppress high frequency noise; and *rate of change*: takes the first derivative. These four operations were chosen as the most important for gross signal conditioning and mapping; a later section addresses filter set extensibility.

Interaction with the filtered signal in the central canvas is analogous to a waveform editor of audio recording software. By default, the canvas shows the latest available data streaming in, with the newest value on the right side. Designers can pause this streaming visualization, scroll through the data, and change how many samples are shown per screen. When fully zoomed out, all the data collected since the beginning of the session is shown.

Demonstration and Mark-up

To begin authoring, the designer performs the action she wants the system to recognize. As an example, to create an interface that activates a light upon firm pressure, the designer may connect a force sensitive resistor (FSR) and press on it with varying degrees of force. In Exemplar, she then marks the resulting signal curve with her mouse. The marked region is highlighted graphically and analyzed as a training example. The designer can manipulate this example region by moving it to a different location through mouse dragging, or by resizing the left and right boundaries. Multiple examples can be provided by adding more regions. Examples can be removed by right-clicking on a region.

In addition to post-demonstration markup, Exemplar also supports real-time annotation through a foot switch, chosen because it leaves the hands free for holding sensors. Using the switch, designers can mark regions and simultaneously work with sensors. Pressing the foot switch starts an example region. The region grows while the switch remains depressed, and concludes when the switch is released. While this technique requires some amount of hand-foot coordination, it enables truly interactive demonstration.

Recognition and Generalization

Recognition works as follows: interactively, as new data arrives for a given sensor, Exemplar analyzes if the data matches the set of given examples. When the system finds a match with a portion of the input signal, that portion is highlighted in the central canvas in a fainter shade of the color than the one used to draw examples. This region grows for the duration of the match, terminating when the signal diverges from the examples.

Exemplar provides two types of matching calculations — thresholds and patterns—selectable as modes for each event (see Figure 2D). With *thresholding*, the minimum and maximum values of the example regions are calculated. The calculation is applied to filtered signals, *e.g.*, it is possible to look for maxima in the smoothed derivative of the input. Incoming data matches if its filtered value falls in between the extrema. *Pattern matching* compares incoming data against the entire example sequence and calculates a distance metric, the extent to which incoming data resembles the example.

Matching parameters can be graphically adjusted through direct manipulation. For threshold events, *min* and *max* values are shown as horizontal lines in the central canvas. These lines can be dragged with the mouse to change the threshold values (see Figure 2G). When parameters are adjusted interactively, matched regions are automatically recalculated and redrawn. Thus, Exemplar always shows how things *would have been classified*. This affords rapid exploration of how changes affect the overall performance of the matching algorithm.

Sensor noise can lead to undesirable oscillation between matching and non-matching states. Exemplar provides three mechanisms for addressing this problem. *First*, a smoothing filter can be applied to the signal. *Second*, the designer can apply hysteresis, or double thresholding, where a boundary is represented by two values which must both be traversed for a state change. Dragging the hysteresis field of a graphical threshold (indicated by "H" in Figure 2G) splits a threshold into two lines. Designers specify the difference between boundary values through the drag distance. *Third*, designers can drag a timeout bar from the right edge of the central canvas to indicate the minimum duration for a matching or non-matching state to be stable before an event is fired.

For pattern matching, Exemplar introduces a direct manipulation technique that offers a visual thresholding solution to the problem of parameterizing the matching algorithm. Exemplar overlays a graph that shows distance between the incoming data and the previously given example on the central canvas's vertical axis. The lower the distance, the better the match. Designers can then adjust a threshold line indicating the *maximum distance* for a positive match. An event is fired when the distance falls below the threshold line. With this technique, the designer's authoring experience is consistent whether applying thresholds or pattern

matching: dragging horizontal threshold bars tunes the specificity of the matching criteria.

Output

Exemplar supports the transformation from sensor input into application events. Exemplar generates two kinds of output events: *continuous* data streams that correspond to filtered input signals, and *discrete* events that fire whenever a thresholding or pattern matching region is found. With these events in hand, the designer then needs to author some behavior, *e.g.*, she needs to specify the application's response to the force sensor push. To integrate Exemplar with other tools, events and data streams can be converted into operating system input events such as key clicks or mouse movements. Injecting OS events affords rapid control over third party applications (cf. [19, 20]). However, injection is relatively brittle because it does not express association semantics (*e.g.*, that the "P" key pauses playback in a video application). For tighter integration with application logic, Exemplar can also be linked to the d.tools authoring environment [19] for rapid off-the-desktop prototypes. Exemplar events are then used to trigger transitions in d.tools' interaction models.

Many Sensors, Many Events

Exemplar scales to more complex applications by providing mechanisms to author *multiple* events for a *single* sensor; to author *individual* events that depend on *multiple* sensors; and to run many independent events simultaneously.

To the right of the central canvas, Exemplar shows a list of event definitions for the currently active sensors (see Figure 2E). Designers can add new events and remove unwanted ones in this view. Each event is given a unique color. A single event from this list is active for editing at a time to keep mouse actions in the central canvas unambiguous: regions drawn by the designer in the canvas always apply to that active event.

Exemplar enables combining sensor data in Boolean AND fashion (*e.g.*, "scroll the map only if the accelerometer is tilted to the left *and* the center button is pressed"). When designers highlight multiple sensor thumbnails, their signals are shown stacked in the central canvas. Examples are now analyzed across all shown sensor signals and events are only generated when all involved sensors match their examples. Boolean OR between events is supported implicitly by creating multiple events. Together, AND/OR combinations enable flexibility in defining events. They reduce — but do not replace — the need to author interaction logic separately.

The authored events for all sensors are always evaluated — and corresponding output is fired — regardless of which sensor is in focus in the central canvas. This allows designers to test multiple interactions simultaneously. To keep this additional state visible, a tree widget shows authored events for all sensors along with their example regions in the lower right corner of the UI (see Figure 2F).

Demonstrate-Edit-Review

The demonstrate-edit-review cycle embodied in Exemplar is an application of the design-test-think paradigm for tools [19, 24]. This paradigm suggests that integrating support for evaluation and analysis into a design tool enables designers to more rapidly gain insight about their project. Exemplar is the first system to apply design-test-think to the domain of sensor data analysis. More importantly, through making demonstrate, edit, and review actions the fundamental authoring operations in the user interface, Exemplar radically shortens iteration times by an order of magnitude (from hours to minutes).

ARCHITECTURE AND IMPLEMENTATION

Exemplar was written using the Java 5.0 SDK as a plug-in for the Eclipse IDE. Integration with Eclipse offers two important benefits: the ability to combine Exemplar with the d.tools prototyping tool to add visual authoring of interaction logic; and extensibility for experts through an API that can be edited using Eclipse's Java tool chain. The graphical interface was implemented with the Eclipse Foundation's SWT toolkit [3].

Signal Input, Output and Display

Consistent with the d.tools architecture, our hardware communicates with Exemplar using OpenSoundControl (OSC). This enables Exemplar to connect to any sensor hardware that supports OSC. In addition to the d.tools I/O board [19], the Exemplar implementation also supports Wiring [8] and Arduino [1] boards with OSC firmware. OSC messages are also used to send events to other applications, *e.g.*, d.tools, Max/MSP, or Flash (with the aid of a relay program). Translation of Exemplar events into system key presses and mouse movements or mouse clicks is realized through the Java Robots package.

Exemplar visualizes up to eight inputs. This number is not an architectural limit; it was chosen based on availability of analog-to-digital ports on common hardware interfaces. Sensors are sampled at 50 Hz with 10-bit resolution and screen graphics are updated at 15-20 Hz. These sampling and display rates have been sufficient for human motion sensing and interactive operation. However, we note that other forms of input, *e.g.*, microphones, require higher sampling rates (8-40 kHz). Support for such devices is not yet included in the current library.

Pattern Recognition

We implemented a Dynamic Time Warping (DTW) algorithm to match demonstrated complex patterns with incoming sensor data. DTW was first used as a spoken-word recognition algorithm [34], and has recently been used in HCI for gesture recognition from sensor data [28]. DTW compares two time-series data sets and computes a metric of the distortion distance required to fit one to the other. It is this distance metric that we visualize and threshold against in pattern mode. DTW was chosen because, contrary to other machine learning techniques, only one training example is required. The DTW technique used in this work is sufficiently effective to enable the interaction techniques

we have introduced. However, we point out that — like related work utilizing machine learning in UI tools [14, 15] —we do not claim optimality of this algorithm in particular.

More broadly, this research—and that of related projects—suggests that significant user experience gains can be realized by integrating machine learning and pattern recognition with direct manipulation. From a developer's perspective, taking steps in this direction may be less daunting than it first appears. For example, Exemplar's DTW technique comprises only a small fraction of code size and development time. We have found that the primary challenge for HCI researchers is the design of appropriate interfaces for working with these techniques, so that users have sufficient control over their behavior without being overwhelmed by a large number of machine-centric "knobs."

Extensibility

While Exemplar's built-in filters are sufficient for many applications, developers also have the option of writing their own filters. Exemplar leverages Eclipse's auto-compilation feature for real-time integration of user-contributed code. This architecture allows engineers on design teams to add to the set of available filters and for users to download filters off the web. Exemplar's filter architecture was inspired by audio processing architectures such as Steinberg's VST [4], which defines a mechanism for how plug-ins receive data from a host, process that stream, and send results back. VST has dramatically expanded the utility of audio-editing programs by enabling third parties to extend the audio-processing library.

EVALUATION

Our evaluation employed a two-pronged approach. First, we applied the Cognitive Dimensions of Notation framework to Exemplar to evaluate its design tradeoffs as a visual authoring environment. Second, we conducted a first-use study in our lab to determine threshold and utility for novices, as well as to assess usability.

Cognitive Dimensions Usability Inspection

The Cognitive Dimension of Notation (CDN) framework [16, 17] offers a high-level inspection method to evaluate the usability of information artifacts. In CDN, artifacts are analyzed as a combination of a *notation* they offer and an *environment* that allows certain manipulations of the notation. CDN is particularly suitable for analysis of visual programming languages. We conducted a CDN evaluation of Exemplar following Blackwell and Green's Cognitive Dimensions Questionnaire [11]. This evaluation offers an analysis of Exemplar according to categories independently identified as relevant, and facilitates comparison with future research systems. We begin with an estimate of how time is spent within the authoring environment, and then proceed to evaluate the software against the framework's cognitive dimensions.

Parts of the System

Exemplar's *main notation* is a visual representation of sensor data with user-generated mark-up. Lab use of Exemplar led us estimate that time is spend as follows:

- **30%** Searching for information within the notation (browsing the signal, visually analyzing the signal)
- **10%** Translating amounts of information into the system (demonstration)
- **20%** Adding bits of information to an existing description (creating and editing mark up, filters)
- **10%** Reorganizing and restructuring descriptions (changing analysis types, re-defining events)
- **30%** Playing around with new ideas in notation without being sure what will result (exploration)

This overview highlights the importance of search, and the function of Exemplar as an exploratory tool.

Dimensions of the Main Notation

Given space constraints, we only present a subset of the 14 CDN dimensions that we deemed most relevant here.

Visibility and Juxtaposability (ability to view components easily): All current sensor inputs are always visible simultaneously as thumbnail views, enabling visual comparison of input data. Viewing multiple signals in close-up is also possible; however, since such a view is exclusively associated with "AND" events combining the shown signals, it is not possible to view independent events at the same time.

Viscosity (ease or difficulty of editing previous work): Event definitions and filter settings in Exemplar are straightforward to edit through direct manipulation. The hardest change to make is improving the pattern recognition if it does not work as expected. Thresholding matching error only allows users to adjust a post-match metric as the internals (the "how" of the algorithm) are hidden.

Diffuseness (succinctness of language): Exemplar's notation is brief, in that users only highlight relevant parts of a signal and define a small number of filter parameters through graphical interaction. However, the notation is not Turing-complete: the set of expressible statements is limited. The length of event descriptions is dependent on the Boolean complexity of the event expressed (how many ORs/ANDs of signal operations there are).

Hard Mental Operations: The greatest mental effort is required to keep track of events that are defined and active, but not visible in the central canvas. To mitigate against this problem, we introduced the *overview list* of all defined interactions (see Figure 2F) which minimizes cost to switch between event views. One important design goal was to make results of operations immediately visible.

Error-proneness (syntax provokes slips): One slip occurred repeatedly in our use of Exemplar: resizing example regions by dragging their boundaries. This was problematic because no visual feedback was given on what the valid screen area was to initiate resizing. Lack of feedback resulted in dupli-

cate regions being drawn, with an accompanying undesired recalculation of thresholds or patterns. Improved input controls on regions can alleviate this problem.

Closeness of Mapping: The sensor signals are the primitives that users operate on. This direct presentation of the signal facilitates consistency between the user's mental model and the system's internal representation.

Role-expressiveness (purpose of a component is readily inferred): Role-expressiveness problems often arise when compatibility with legacy systems is required. Since Exemplar was designed from scratch for the express purpose of viewing, manipulating and marking up signals, this it not a concern. While the result of applying operations is always visible, the implementation "meaning" of filters and pattern recognition is hidden.

Secondary Notations: Currently, Exemplar permits users to label events, but not filter settings or signal regions. As comments allow users to capture information outside the "syntax" of Exemplar, this is an area for future work.

Progressive Evaluation: Real-time visual feedback enables fluid evaluation of the state of an interaction design at any point. Furthermore, Exemplar sessions can be saved and retrieved for subsequent sessions through disk serialization.

In summary, Exemplar performs well with respect to visibility, closeness of mapping, and progressive evaluation. Many of the identified challenges stem from the difficulties of displaying multiple sensor visualizations simultaneously. These can be addressed through interface improvements—they are not intrinsic shortcomings of the approach.

First-use Study

We conducted a controlled study of Exemplar in our laboratory to assess the ease of use and felicity of our tool for design prototyping. The study group comprised twelve participants. Ten were graduate students or alumni of our university; two were undergraduates. While all participants had some prior HCI design experience, they came from a variety of educational backgrounds: four from Computer Science/HCI, four from other Engineering fields, two from Education, and two from the Humanities. Participants' ages ranged from 19 to 31; five were male, seven female. Two of the twelve participants served as pilot testers. Eight participants had had some prior exposure to sensor programming, but none self-reported to be experts (see Figure 4).

Study Protocol

Participants were seated at a dual-screen workstation with an Exemplar hardware interface. The Exemplar software was shown on one screen; a help document on sensors was shown on the other. Participants were asked to author interactions that controlled graphics on a large projection display (see Figure 5). We chose this large display to encourage participants to think beyond the desktop in their designs. We chose graphical instead of physical output since our study focused on authoring responses to sensor input, not on actuation.

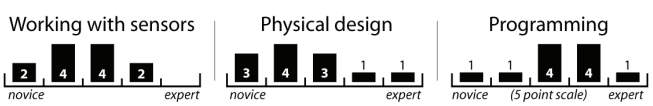

Figure 4. Prior experience of our study participants.

Figure 5. Physical setup of the first-use study.

Individual study sessions lasted two hours. Sessions started with a demonstration of Exemplar. We also introduced the set of available sensors, which comprised buttons, switches, capacitive touch sensors, light sensors, infrared distance rangers, resistive position sensors, force-sensitive resistors (FSRs), load cells, bend sensors, 2D joysticks, and 3D accelerometers. Participants were given three design tasks. For each, we provided a mapping of triggers available in Exemplar to behaviors in the instructions (*e.g.*, sending an event called "hello" activated the display of the *hello* graphic in the first task).

The first task was a "Hello World" application. Subjects were asked to display a *hello* graphic when a FSR was pressed (through thresholding) while independently showing a *world* graphic when a second FSR was pressed three times in a row (through pattern recognition).

The second task required participants to augment a bicycle helmet with automatic blinkers such that tilting the helmet left or right caused the associated blinkers to signal. Participants had to attach sensors to and test with a real helmet. Blinker output was simulated with graphics on a "mirror" on the projection display. This task was inspired by Selker *et al.*'s Smart Helmet [35].

The last task was an open-ended design exercise to author new motion-based controls for at least one of two computer games. The first was a spaceship navigation game in which the player has to keep a ship aloft, collect points and safely land using three discrete events to fire thrusters (up, left, and right). The second game was a shooting gallery with continuous X/Y position control used to aim, and a discrete trigger to shoot moving targets.

Study Results

In our post-test survey, participants ranked Exemplar highly for decreasing the time required to build prototypes compared to their prior practice (mean=4.8, median=5 on a 5-point Likert scale, σ=0.42); for facilitating rapid modification (mean=4.7, median=5, σ=0.48); for enabling them to experiment more (mean=4.7, median=5, σ=0.48); and for helping them understand user experience (mean=4.3, median=4; σ=0.48). Responses were less conclusive on how use of Exemplar would affect the number of prototypes built, and whether it helped focus or distracted from design details (σ>1.0 in each case). Detailed results are presented in Figure 6.

Successes

All participants successfully completed the two first two tasks and built at least one game controller. The game controller designs spanned a wide range of solutions (see Figure 7). Once familiar with the basic authoring techniques, many participants spent the majority of their time sketching and brainstorming design solutions, and testing and refining their design. In aggregate, implementation composed less than a third of their design time. This rapid iteration enabled participants to explore up to four different control schemes for a game. We see this as a success of enabling epistemic activity: participants spent their time on design thinking rather than implementation tinkering.

Exemplar was used for exploration: given an unfamiliar sensor, participants were able to figure out how to employ it for their purposes. For example, real-time feedback enabled participants to find out which axes of a multi-axis accelerometer were pertinent for their design. Participants also tried multiple sensors for a given interaction idea to explore the fit between design intent and available technologies.

Interestingly, performance of beginners and experts under Exemplar was comparable in terms of task completion time and breadth of ideation. Two possible explanations for this situation are that either Exemplar was successful in lowering the threshold to entry for the types of scenarios tested, or that it encumbered experts from expressing their knowledge. The absence of complaints by experts in the post-test surveys provides some support for the first hypothesis.

Shortcomings discovered

Participants identified two key areas for improvement. One recurring theme in the feedback was the need for visualization of hidden state. At the time of the study, participants could only see events that corresponded to the sensor in focus. Although other events were still active, there was no comprehensive way of listing them. Also, the highlighted regions corresponding to training examples were hard to retrieve after time had passed because the example regions were pushed too far off-screen into the history. To address these difficulties, Exemplar now displays a full list of active

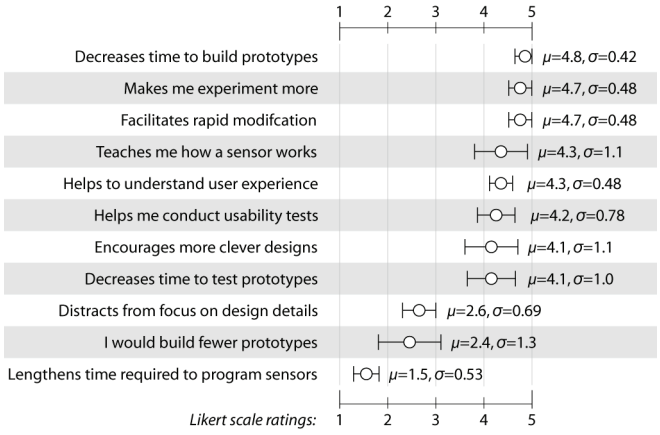

Figure 6. Post-experiment questionnaire results. Error bars indicate ½ standard deviation in each direction.

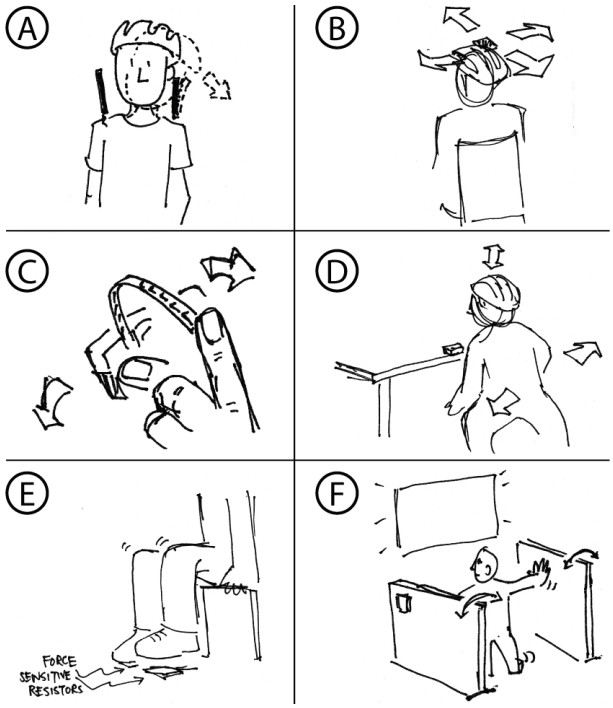

Figure 7. Interaction designs from the user study. A: turning on blinkers by detecting head tilt with bend sensors; B: accelerometer used as continuous 2D head mouse; C: aiming and shooting with accelerometer and bend sensor; D: navigation through full body movement; E: bi-pedal navigation through FSRs; F: navigation by hitting the walls of a booth (vibration sensed with accelerometers).

events, along with the corresponding example regions. Selecting those regions jumps to the time of their definition in the central canvas.

Expert users expressed a need for finer control over hysteresis parameters for thresholding, and a visualization of time and value units on the axes of the signal display. In response, we added direct manipulation of hysteresis and timeout parameters to threshold events.

The importance of displaying numerical data in the visualization to aid understanding of signal properties deserves further study. Participants also requested ways to provide negative examples, techniques for displaying multiple large sensor visualizations simultaneously, and finer control over the timing for pattern matching (in terms of both latency and duration). We leave these to future work.

RELATED WORK

The research embodied in Exemplar was directly motivated by our experience with d.tools [19], which introduced a *value entry by demonstration* technique: in a visualization of a single sensor's signal, a user could copy the latest value into the threshold property of a state transition with a key press. Exemplar extends the d.tools work by introducing:

- Direct manipulation techniques to control generalization from examples.
- Pattern matching for complex signals with graphical editing of matching criteria.

- Graphical feedback of matching criteria for sensor histories.
- A raised ceiling by working with multiple sensors, multiple events, and extensible filters.
- Evaluation through a lab study and CDN analysis.

Exemplar also relates to three larger areas of work: research into programming by demonstration for ubiquitous computing, tools for musical controller design, and signal processing and analysis software. We discuss each in turn.

Ubicomp PBD

The closest predecessor to Exemplar in approach and scope is a CAPella [14]. This system focused on authoring binary context recognizers by demonstration (*e.g.*, is there a meeting going on in the conference room?), through combining data streams from discrete sensors, a vision algorithm, and microphone input. Exemplar shares inspiration with a CAPella, but it offers important architectural contributions beyond this work. First, a CAPella was not a real-time interactive authoring tool: the authors of a CAPella reported the targeted iteration cycle to be on the order of days, not minutes as with Exemplar. Also, a CAPella did not provide strong support for continuous data. More importantly, a CAPella did not offer designers control over how the generalization step of the PBD algorithm was performed beyond marking regions. We believe that this limitation was partially responsible for the low recognition rates reported (between 50% and 78.6% for binary decisions).

We also drew inspiration for Exemplar from Fails and Olsen's interaction technique for end-user training of vision recognizers, Image Processing with Crayons [15]. It enables users to draw on training images, selecting image areas (*e.g.*, hands or note cards) that they would like the vision system to recognize. Crayons complements our work well, offering a compelling solution to learning from independent images, where as Exemplar introduces an interface for learning from time-series data.

Monet [25] is a sketch-based prototyping system for animating hand-drawn widgets. Monet learns geometric transformations applied to widgets through mouse movement. Monet is a GUI-centric application of PBD that supports both continuous and discrete outputs. It uses a different mathematical approach (continuous function approximation using radial basis functions centered on screen pixels).

Papier-Mâché [23], while not a PBD system, also makes physical input processing more accessible. It supports how designers specify input of interest—its main insight was to create software event abstractions that were common across a set of input technologies. This enables designers to rapidly exchange input technologies. Papier-Mâché focused on identity-based input and discrete events.

Tools for musical controller design

Design tools for electronic musical instruments also address the issue of mapping continuous sensor input. There are two important differences between using sensors as inputs for user interfaces and using sensors as controls for real-time music synthesis: musicians are often concerned with continuous-to-continuous mappings, and the frequency of the output signal (the music) is much greater than that of the input signal (the performer's actions). Exemplar focuses on sparser output events that are frequently of discrete nature.

Steiner's [hid] toolkit [37], is a set of objects, called "externals," that support USB Human Interface Device input in Pd [32], the visual multimedia programming environment. The toolkit does not focus on the interaction design for exploring and specifying signal transformations. It uses Pd's existing object architecture and does not offer example-based learning. MnM [10], also an extension for Pd and Max/MSP, focuses on advanced mathematical signal transformations. It aims to make complex operations accessible to those already familiar with sensor interaction design but does not integrate exploration, design, and test functions.

FlexiGesture is an electronic instrument that can learn gestures to trigger sample playback [28]. It embodies programming by demonstration in a fixed form factor. Users can program which movements should trigger which samples by demonstration, but they cannot change the set of inputs. Exemplar generalizes FlexiGesture's approach into a design tool for variable of input and output configurations. We share the use of the DTW algorithm for pattern recognition with FlexiGesture.

Electrical Engineering Signal Analysis

Digital Signal Processing (DSP) software packages such as LabView [2] are the state-of-the art engineering tools for working with sensors. LabView and Exemplar support different user populations, with different respective needs and expectations. LabView offers a sophisticated Turing-complete visual programming language that allows professional users to create dataflow algorithms and signal visualizations. LabView's focus on measuring, analyzing, and processing signals supports professional engineers and trades off generality for a high threshold for use. In contrast, Exemplar integrates a subset of the DSP functions LabView offers into an integrated design environment that encourages exploration and rapid iteration by designers.

CONCLUSION

This paper introduced techniques for authoring sensor-based interactions through programming by demonstration, where the crucial generalization step is user-editable through direct manipulation of thresholding and pattern matching. Future work consists of investigating how interactive editing can be extended to other time-series signals and different kinds of matching strategies. Exemplar has been released as open source under the BSD license.

ACKNOWLEDGMENTS

We thank Wendy Ju for illustrating Figures 5 and 7, Intel for donating PCs, and MediaX/DNP for funding.

REFERENCES

1 *Arduino Physical Computing Platform*, 2006. http://www.arduino.cc

2 *LabView*, 2006. National Instruments. http://www.ni.com/labview

3 *SWT: The Standard Widget Toolkit*, 2006. The Eclipse Foundation. http://www.eclipse.org/swt

4 *Virtual Studio Technology (VST)*, 2006. Steinberg. http://steinbergcanada.com/technology/vst.htm

5 Andreae, P., *Justified Generalization: Acquiring Procedures from Examples*, Unpublished PhD Thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1985.

6 Avrahami, D. and S. E. Hudson. Forming interactivity: A tool for rapid prototyping of physical interactive products. In Proceedings of *DIS: ACM Conference on Designing Interactive Systems*. pp. 141–46, 2002.

7 Ballagas, R., M. Ringel, M. Stone, and J. Borchers. iStuff: A physical user interface toolkit for ubiquitous computing environments. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 537–44, 2003.

8 Barragan, H., *Wiring: Prototyping Physical Interaction Design*, Interaction Design Institute Ivrea, Italy, 2004.

9 Bellotti, V., M. Back, W. K. Edwards, R. E. Grinter, A. Henderson, and C. Lopes. Making sense of sensing systems: five questions for designers and researchers. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 415–422, 2002.

10 Bevilacqua, F., R. Müller, and N. Schnell, MnM: a Max/MSP mapping toolbox. In Proceedings of *NIME: Conference on New Interfaces for Musical Expression*. pp. 85–88, 2005.

11 Blackwell, A. and T. Green, *A Cognitive Dimensions Questionnaire*, 2000. http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDquestionnaire.pdf

12 Card, S. K., J. D. Mackinlay, and G. G. Roebertson. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Systems*. **9**(2): ACM Press. pp. 99–122, 1991.

13 Cypher, A., ed. *Watch What I Do - Programming by Demonstration*. MIT Press: Cambridge, MA. 652 pp., 1993.

14 Dey, A. K., R. Hamid, C. Beckmann, I. Li, and D. Hsu. a CAPpella: Programming by demonstration of context-aware applications. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. pp 33–40, 2004.

15 Fails, J. and D. Olsen. A design tool for camera-based interaction. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. pp 449–456, 2003.

16 Green, T. R. G. Cognitive dimensions of notations. *People and Computers V*. pp. 443-60, 1989.

17 Green, T. R. G. and M. Petre. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing* **7**(2). pp. 131–74, 1996.

18 Greenberg, S. and C. Fitchett. Phidgets: Easy development of physical interfaces through physical widgets. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*. pp. 209–18, 2001.

19 Hartmann, B., S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher and J. Gee. Reflective physical prototyping through integrated design, test, and analysis. In Proceedings of *UIST 2006: ACM Symposium on User Interface Software and Technology*. pp. 299–308, 2006.

20 Hudson, S. and J. Mankoff. Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*. pp 289–298, 2006.

21 Hutchins, E. L., J. D. Hollan, and D. A. Norman. Direct Manipulation Interfaces. *Human-Computer Interaction* **1**(4). pp. 311–38, 1985.

22 Klemmer, S. R., B. Hartmann, and L. Takayama. How Bodies Matter: Five Themes for Interaction Design. In Proceedings of *DIS: ACM Conference on Designing Interactive Systems*. pp. 140–149, 2006.

23 Klemmer, S. R., J. Li, J. Lin, and J. A. Landay. Papier-Mâché: Toolkit Support for Tangible Input. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 399–406, 2004.

24 Klemmer, S. R., A. K. Sinha, J. Chen, J. A. Landay, N. Aboobaker, and A. Wang. SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*. pp. 1–10, 2000.

25 Li, Y. and J. A. Landay. Informal prototyping of continuous graphical interactions by demonstration. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*. pp. 221–30, 2005.

26 Lieberman, H., ed. *Your Wish is my Command*. Morgan Kaufmann. 416 pp., 2001.

27 Lieberman, H., F. Paternò, and V. Wulf, ed. *End-User Development*. Springer: Dordrecht, Netherlands. 492 pp., 2006.

28 Merrill, D. J. and J. A. Paradiso, Personalization, Expressivity, and Learnability of an Implicit Mapping Strategy for Physical Interfaces. In Extended Abstracts of *CHI: ACM Conference on Human Factors in Computing Systems*. 2005.

29 Munch, S., J. Kreuziger, M. Kaiser, and R. Dillmann. Robot programming by demonstration – using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. In Proceedings of *International Symposium on Industrial Robots*. 1994.

30 Nardi, B. A., *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA: MIT Press. 1993.

31 Norman, D. A., *Things that make us smart: Defending human attributes in the age of the machine*: Addison Wesley Publishing Company. 290 pp., 1993.

32 Puckette, M., *Pd*, 2006. http://crca.ucsd.edu/~msp/software.html

33 Rogers, Y. and H. Muller. A framework for designing sensor-based interactions to promote exploration and reflection in play. *International Journal of Human-Computer Studies* **64**(1). pp. 1–14, 2006.

34 Sakoe, H. and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **26**(1): IEEE. pp. 43–49, 1978.

35 Selker, T., *A bike helmet built for road hazards*, 2006. http://news.com.com/2300-1008_3-6111157-2.html

36 Shneiderman, B., Overview + Detail, in *Readings in Information Visualization*, S.K. Card, J.D. Mackinlay, and B. Shneiderman, ed. Morgan Kaufmann, 1996.

37 Steiner, H.-C., [hid] toolkit: a unified framework for instrument design. In Proceedings of *NIME: Conference on New Interfaces for Musical Expression*. pp. 140-143, 2005.

38 Tufte, E. R., *Envisioning Information*. Cheshire, CT: Graphics Press LLC. 126 pp. 1990.