Sketching Interactive Systems with Sketchify

ŽELJKO OBRENOVIC and JEAN-BERNARD MARTENS, Eindhoven University of Technology

Recent discussions in the interaction design community have called attention to sketching as an omnipresent element of any disciplined activity of design, and have pointed out that sketching should be extended beyond the simple creation of a pencil trace on paper. More specifically, the need to deal with all attributes of a user experience, especially the timing, phrasing, and feel of the interaction, has been identified. In this article, we propose extending the concept of sketching with a pencil on paper to the more generic concept of fluent exploration of interactive materials. We define interactive materials as any piece of software or hardware that represents or simulates a part of the interactive user experience, such as input from sensors, output in the form of sound, video, or image, or interaction with Web services or specialized programs. We have implemented the proposed concept within Sketchify, a tool for sketching user interfaces. Sketchify gives designers the freedom to manipulate interactive materials by combining elements of traditional freehand sketching with functional extensions and end-user programming tools, such as spreadsheets and scripting. We have evaluated Sketchify in the education of interaction designers, identifying both successful aspects and aspects that need further improvements.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*; H.1.2 [**Models and Principles**]: User/Machine Systems

General Terms: Design, Theory

Additional Key Words and Phrases: Sketching, interaction design, user interface software tools, design process, rapid prototyping

ACM Reference Format:
Obrenovic, Ž. and Martens, J.-B. 2011. Sketching interactive systems with sketchify. ACM Trans. Comput.-Hum. Interact. 18, 1, Article 4 (April 2011), 38 pages.
DOI = 10.1145/1959022.1959026 http://doi.acm.org/10.1145/1959022.1959026

1. INTRODUCTION

Sketching is at the heart of design. Many studies of the design practice, such as recent contributions from Buxton [2007], Krippendorff [2006], and Moggridge [2007], have called attention to sketching as an omnipresent element of any disciplined activity of design. Disciplines such as graphical design and architecture can boast a rich tradition in sketching, and offer courses to students in order to improve their sketching skills. However, for interaction designers who want to design new user interfaces, existing sketching techniques are too limited. Buxton has argued that while it is relatively easy to sketch the physical shape of an interaction device or the graphical layout of a user interface, interaction designers lack tools that enable them to sketch the dynamics of the interaction, let alone the overall user experience [Buxton 2007]. For example, pencil

This work was supported by the Innovation-Oriented Research Programme 'Integral Product Creation and Realization (IOP IPCR)' of the Netherlands Ministry of Economic Affairs, Agriculture and Innovation.
Authors' addresses: Ž. Obrenovic (corresponding author) and J.-B. Martens, Eindhoven University of Technology, The Netherlands; email: z.obrenovic@tue.nl.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2011 ACM 1073-0516/2011/04-ART4 $10.00
DOI 10.1145/1959022.1959026 http://doi.acm.org/10.1145/1959022.1959026

and paper provide few means to sketch speech interaction, or to illustrate interaction scenarios in domains such as ambient intelligence, tangible interaction, multimodal interaction, or pervasive computing.

The identified issue of lack of tools for sketching is also confirmed by our own experience in the education of students of interaction design. Based on these practical experiences and on our understanding of existing theoretical contributions, we introduced a novel approach and tool for sketching, adopting two main premises.

—The primary objective of interaction designers in the early design stage is clarifying the user experience and the associated user-system interaction [Moggridge 2007]. Interaction designers (and students) need better techniques than those currently available for sketching such experiences and interactions. As in the case of traditional sketching, these techniques need to combine speed and freedom of expression, and need to assist in producing an output that invites discussion.
—Sketching should be extended beyond the simple creation of a pencil trace on paper to deal with important attributes of the overall user experience, especially time, phrasing, and feel [Buxton 2007]. Some aspects of specifying interactive system behavior are beyond freehand drawings and we need tools that can seamlessly integrate sketching with more traditional (end-user) programming techniques.

To support and explore our view, we have developed Sketchify, a tool that implements our extension of paper and pencil sketching to the more generic concept of *fluent exploration of interactive materials*. Interactive materials can be any piece of software or hardware that represents or simulates a part of the interactive user experience, such as input from sensors, output in the form of audio, video, or drawings, or interaction with Web services. Through the manipulation of interactive materials, designers create *interactive sketches*, which are rough illustrations of the interaction scenarios or interaction techniques that they have in mind. With our tool designers can, for instance, combine elements of freehand sketching with end-user programming, such as spreadsheets or scripting, needed in order to create an intelligent system behavior.

Figure 1 provides a very simple example of an “interactive sketch” created with our tool. This example was created by one of our students to illustrate the working of an “intelligent window,” where a user can see what is going on in another room by “cleaning” the window with a hand gesture. To create this sketch, a motion detector is used, where the intensity of motion is mapped to the transparency of the image that represents the window. This example illustrates several important aspects of our tool. Firstly, a sketch, as we define it, need not be restricted to a drawing, but can incorporate any component that helps the designer to develop and show his idea about interaction, provided that this can be realized in a quick and timely fashion, and that such a sketch is inexpensive, disposable, and doesn’t contain unnecessary details. Secondly, our tool allows the designer to *sketch the experience* that some interaction scenario will bring to a user by enabling a user, as well as the designer himself, to immediately *try out the intended interaction*. While a drawing can, through lines and text, illustrate how the interaction will occur, it is more insightful and convincing to actually experience how the transparency of the image changes, that is, how the window “opens,” as a result of the user gesture. Lastly, we are primarily interested in supporting interaction designers whose main objective is to design the *dynamics of the interaction* [Moggridge 2007]. The image in our example is secondary; what is important is the *change* of that image. In other words, in this example the designer is primarily interested in the development of the mapping from the user gesture to the image transparency as this, rather than the image itself, may be expected to have a major influence on the user experience.

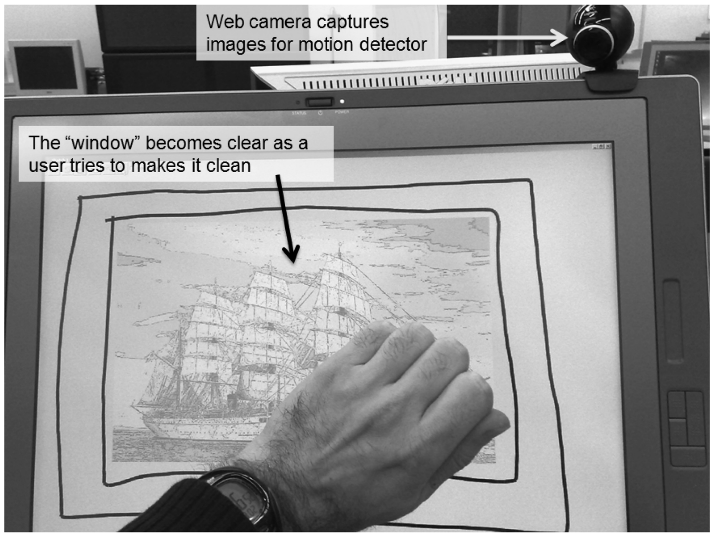

Web camera captures images for motion detector
The "window" becomes clear as a user tries to makes it clean

Fig. 1. In this example, a camera-based motion detector is used to estimate the intensity of the hand motion. The transparency of the window changes in response to the estimated motion intensity.

In the next section we introduce an example scenario to clarify how we envision sketching in the context of designing novel interactive systems. Next, we present some existing software tools for sketching and prototyping and discuss why, in our view, they offer only partial solutions to the requirements imposed by this scenario. From Section 4 onwards we introduce Sketchify, our toolset for sketching interactive systems. We describe its architecture and its implementation, discuss its benefits and limitations, and compare it with existing solutions. We conclude with a summary of our contributions and with plans for future work.

To improve readability of the text, we have moved most implementation details into four online Appendices (A, B, C, and D). For readers interested in trying out Sketchify, we provide an open-source version of the program, together with some introductory videos.[1]

2. EXAMPLE SCENARIO

To illustrate our vision about sketching in the context of the design of interactive systems, we introduce an example scenario, with Anne in the role of a student of interaction design. Our scenario is based on an analogy to "traditional" paper and pencil sketching, where a designer quickly draws up ideas, reflects on them and makes changes, creates variations, and discusses them with colleagues.

Anne is a student at the department of Industrial Design, and she is working on her final bachelor project. Her goal is to design an "intelligent coffee machine": ICoM, a context-aware system that, in addition to making hot drinks, senses the presence of people, and has secondary functions such as providing ambient music and lighting. She has already done user surveys, and has identified, in general terms, the desired functionality of the system. Specifically, she has concluded that ICoM should support the following functions:

—Detect the presence of users, and classify their distance from ICoM in two categories. When one or more persons are within the vicinity of the machine, the system should

[1]http://sketchify.sf.net/

"express interest" in these people, and when a person comes close to the machine, the system should "become serious" and offer a menu with available hot drinks.
—When no one is close to the system, it should switch into behaving as an "ambient box," providing a discrete lighting and music.

Anne now starts to define the behavior that ICoM should have in order to support the identified functionality. She opens her interactive "sketchpad" and starts to "sketch" elements for the first part of the functionality. Her sketchpad enables her to easily and quickly explore and work with various devices and sensors. Anne does not have a lot of experience in working with sensing equipment, but she is aware of the general possibilities offered by sensors, and that detecting the presence of people, which is required for her scenario, can be achieved in several ways. She, for example, could consider using RFID sensors, but this would require users to wear RFID tags, which is probably unrealistic, so she rejects that option. Another possibility would be to use pressure sensors on the floor or a camera-based computer-vision sensor. Her sketching tool, for example, offers her an option to integrate a Wii Fit pressure sensor.[2] She decides to play with this option, and borrows the Wii Fit Balance Board device from the faculty service desk. She connects the device to her computer through a Bluetooth link, and starts the Wii module in the sketchpad. Her sketching environment offers her a simple interface towards the device and various ways to control and visualize data coming from and going to the device. By inspecting this interface, she sees that, amongst others, the Wii Fit device updates four variables, each representing different areas in which the user may be standing. She tries the device herself, observing how the data changes when she stands on it. This immediately gives her another idea: the Wii Fit does not only detect the presence of people in front of the ICoM, but also potentially allows alternative ways of interacting, by stepping or balancing on different parts of the board. Using the freehand sketching extension of her tool, she quickly creates several drawings representing various screens of the coffee machine, such as an entry screen, and a screen for selecting the coffee type and amount of sugar. Her environment enables her to define transitions between these drawings as a function of sensor variables, and she quickly creates a simple interactive sketch, where a user can select a coffee type and amount of sugar by balancing on the Wii Fit board.

She saves the last sketch, and explores some other options for detecting the presence of people. She attaches a Web camera to her computer, and tries camera-based motion and face detection. The face detector seems to be more promising, as it can tell how many people are in front of the screen, and the face size can roughly indicate how close they are to the machine. As the face detector does not tell Anne the distance of people directly, she has to derive it. Her environment offers her a variety of programming options, but as she is not an experienced programmer, she decides to use a spreadsheet and to import the variables for face size within it, estimating the distance by dividing the face size by some factor. To obtain this factor, she simply tries which face size the sensor gives when she stands at different distances from the Web camera. She now reuses some freehand drawings from her first sketch, and creates an interactive scenario, where different screens appear depending on the estimated distance of the user. When the user is close enough he can select drinks and other options by clicking on them. She also creates a version where options can be selected by means of head motions, but quickly sees that this is not an intuitive way of interaction, and does not pursue it further.

To add more diversity to her ideas, she also creates several versions of the navigation interface, creating one version where a user selects hot drink options by speech, using

[2]http://www.nintendo.com/wiifit/

a speech recognizer and a text-to-speech engine available within the sketchpad. She also creates a version of her sketches that connects to the Google News search Web service, so that the user can read the news on the ICoM display while waiting for a drink to be prepared.

To define the second part of the functionality, where ICoM works in the "ambient box" mode, she uses the MP3 player extension available within her sketching platform, and creates a simple timer that starts the playback of a music item in a list of MP3 files after some predefined time of inactivity.

Now having several sketches and variations of her ideas ready, she calls some of her colleagues to try out and discuss these "sketches" before she consolidates her design decisions and starts to create more advanced and polished prototypes with which she intends to test the usability and user appreciation of her ideas.

3. EXISTING SOLUTIONS

Our example scenario illustrates the need for a simple yet powerful design environment that can offer integrated use of various elements, including sensing devices, graphical editors, and Web services. In this section, we describe some existing solutions, and discuss their possibilities as well as their limitations when it comes to supporting applications such as the one described in the example scenario. We start with sketching on paper, paper prototyping, and screen prototyping, before proceeding to more complex solutions such as electronic sketching and platforms for the rapid prototyping and programming of interactive systems.

3.1. Sketching on Paper, Paper Prototyping, and Screen Prototyping

To illustrate her ideas about the interaction with the coffee machine, our protagonist Anne could have created freehand drawings. As most students, Anne has developed her drawing skills through courses offered at the department of industrial design. Such courses also cover aspects of communicating the dynamics of interaction by means of graphical elements such as arrows, textual annotations, or comic-like sequences of images [Olofsson and Sjölén 2005]. Existing software programs such as Photoshop, Painter, or Gimp can assist Anne in this task, as they support free-form sketching using metaphors based on conventional tools, such as, pen, pencil, eraser, or brush.

She could also extend her drawings towards paper prototypes, which have been used successfully in the design of many interactive products, including computer-based applications, mobile devices, and Web sites [Grady 2000; Rettig 1994; Snyder 2003]. In such prototypes, all elements of the interface are sketched, and arrows are used to connect the screens and to communicate the interaction paths that result from a user activating specific interaction elements [Pering 2002].

Anne could make her drawings more interactive using screen prototyping techniques, importing, for example, her images into tools such as Microsoft PowerPoint,[3] Balsamiq,[4] Pencil,[5] or OmniGraffle,[6] and animating series of screens to simulate possible paths of interaction (we discuss more advanced programming techniques, such as Flash, in Section 3.3).

Paper sketches and screen prototypes can be created with ease and they can be very effective in a number of situations. However, they can help Anne only in a limited way. She can easily create the graphical elements of her solutions, such as the shapes that will be shown on the screen, and she can simulate the sequence of these paper sketches,

[3]http://office.microsoft.com/powerpoint/
[4]http://www.balsamiq.com/
[5]http://www.evolus.vn/Pencil/
[6]http://www.omnigroup.com/applications/OmniGraffle/

but overall interaction, where sensing devices and the dynamics of the responses are also taken into account, can be described only in very abstract terms. Moreover, as Anne is not experienced in working with sensing technologies, paper sketching does not allow her to explore the possibilities and limitations of such technologies. Through exploring technologies, she can get more concrete ideas about how they may be employed best. In this respect paper-based sketches cannot help her either.

3.2. Electronic Sketching Systems

In contrast to paint programs, where sketching is used to create images, electronic sketching systems let the user sketch using an electronic pad and stylus and interpret the user's strokes in order to create a semantic representation of the sketch [Igarashi and Zeleznik 2007]. Computer graphics researchers have developed a range of such systems. Starting from Sutherland's seminal work on Sketchpad [Sutherland 1963], several pen-based systems with varying target domains have been proposed to date. Some examples include SketchCAD [Kara and Shimada 2007], a system for the rapid creation of free-form curves and surfaces, SketchIT [Stahovich 1998], a system for creating technical drawings, the SKETCH system for sketching 3D scenes [Zeleznik et al. 1996], and the "sketching reality" system, used for converting freehand sketches into realistically looking models [Chen at al. 2008]. Systems such as Teddy [Igarashi et al. 1999], Vteddy [Owada et al. 2003], SmoothTeddy [Igarashi and Hughes 2003], and ShapeShop [Schmidt et al. 2005] allow creating even more complex 3D shapes, such as shapes of animals or human anatomy. Gestures can also be used to define animation, such as in the system for articulated figure animation [Davis et al. 2003], the Motion Doodles system [Thorne et al. 2004], or the K-Sketch system [Davis et al. 2008].

In the domain of user interfaces, there exist several similar systems. SILK [Landay 1996] is an electronic sketching tool for the early design of graphical WIMP-based user interfaces. SILK enables a designer to draw graphical interface elements and attempts to recognize widgets in the sketch, automatically generating a default behavior for the recognized widgets. Using SILK, a user interface designer can create storyboards to illustrate transitions between sketches. DENIM [Lin et al. 2000] is another electronic sketching tool aimed at supporting early Web interface design. In a similar way as with the storyboards in SILK, a designer can sketch navigational links from source widgets to destination pages. DEMAIS is a multimedia sketch-based editor [Bailey et al. 2001], which, in addition to structuring pages and defining the navigation structure, also enables the use of dynamic media such as audio, video, and animation. The system includes a sketch-based, interactive multimedia storyboard tool through which behavior can be quickly edited using gestures that are part of an expressive visual language.

Sketch-based systems are a promising new direction for design tools, enabling designers to create interactive systems with ease, using intuitive and natural pen gestures. The drawback of sketch-based systems, from the viewpoint of our example scenario, is that all of the described electronic sketching tools are specialized and domain specific, and have been successfully used only in inherently graphical domains that have a stable and well-known set of primitives, such as 2D and 3D graphics, WIMP interfaces, or Web sites. Anne, for example, could use DENIM to create the forms and the transitions between the forms that take place when a customer is making his coffee selection, but for other elements of her solution she would have to resort to alternative tools.

3.3. Platforms for Rapid Prototyping of Interactive Systems

Another path that Anne could pursue is to actually try to implement a simplified version of her system using rapid prototyping and development tools.

Some existing low-fidelity prototyping environments provide ways to quickly create prototypes where inputs can be taken from external buttons or sensors. Examples include Switcharoo for physical interactive products [Avrahami and Hudson 2002], Calder and Phidgets[7] for physical interfaces [Greenberg and Boyle 2002; Lee et al. 2004; Greenberg and Fitchett 2001], Buck prototyping for mobile devices [Pering 2002], rapid prototyping for mobile devices using augmented reality technology [Nam and Lee 2003], DART for augmented reality systems [MacIntyre et al. 2004], d.tools for physical prototyping [Hartman et al. 2006], Topiary for prototyping of location-enhanced applications [Li et al. 2004], Outpost [Klemmer et al. 2001], and Activity Studio for prototyping of ubicomp applications [Li and Landay 2008].

These low-fidelity prototyping environments may be an excellent choice for exploration of interactions in various domains. The domain that Anne is addressing, however, somehow crosses these domains, and requires a variety of sensory inputs and links to outside services.

There are more professionally integrated environments that can be used to develop complex interactive applications. For example, Max/MSP[8] and the family of related patcher programming languages such as Pd, Max/FTS, ISPW Max, Max/MSP, or jMax, provide a graphical development environment for music and multimedia. The Max program, for example, is highly modular, with most routines existing in the form of shared libraries. Through these libraries, various input and output modules can be used. There also exist research platforms such as EyesWeb[9] that support the development of real-time multimodal interactive applications, especially those using expressive gestures. OpenInterface[10] is another such platform, aimed at a component-based development of multimodal applications. These systems can enable a designer to define a range of effects using an easy to understand flow-chart metaphor. However, such systems limit a designer in several ways. While they may be efficient to use in specific domains, such as music or video, their usage in other domains may not be straightforward. In addition, they often require too much precise specification, partly due to the fact that they are primarily developed for advanced prototyping rather than for sketching.

Anne could consider general -purpose programming languages, that is, higher-level languages such as Flash or Processing, or fully featured languages such as Java or C++. In hands of a skilled programmer such languages are powerful tools, and they provide lots of support and libraries for implementing all aspects of our interaction scenario. Programming, however, is usually not appropriate in the early stages of development, and most interaction designers are not skilled programmers in the first place. Even if Anne were an experienced developer, programming of her system would still require significant time and effort. Such investment is simply too high for the intended purpose, which is the generation of new ideas and the exploration of interaction possibilities.

3.4. Summary of the Limitations of Existing Solutions

Existing solutions provide a broad set of possibilities for development of interactive solutions, and they introduce a range of new ideas and inspirations for development of new design tools. Many of these ideas have significantly influenced and inspired our approach. However, having in mind our goals, and summarizing our overview of the existing solutions, we can make the following observations.

[7]http://www.phidgets.com/
[8]http://www.cycling74.com/
[9]http://www.infomus.org/EyesWeb/EywPlatform.html
[10]http://www.openinterface.org/platform/

—*Lack of (integrated) tools*: there are currently no tools capable of supporting the diversity of technologies and design issues required in our example scenario, especially not if we add the requirement that such tools need to support rapid, sketch-like interaction.
—*Specialization and limited extensibility of tools*: there are lots of specialized tools and pieces of software that can cover aspects of the desired sketching functionality, but they cannot be used easily in an integrated way. Extending a particular tool towards using it in another domain can be very time consuming and expensive and is moreover not always feasible (especially for tools that are not open source).

We can add two additional factors that influence and limit the broader adoption of existing tools by interaction designers.

—*The diversity of users*: interaction designers have very diverse backgrounds and expertise, and most of them are not developers or programmers. Many existing tools require a level of expertise that goes beyond what can be expected of most interaction designers. As more and more (industrial and graphical) designers are entering the field of interaction design, this aspect can be expected to become increasingly important. This can results in better and more compelling systems and interactions, as diverse designers can bring into a design process unique expertise and insights, but we need to provide them with appropriate technologies to effectively prototype and interactively sketch their ideas.
—*Technology evolution*: interaction design is a domain where the technological base is changing rapidly, and designers need constantly to learn new technologies and tools. One of the consequences is that even if we create a design tool that can address all identified technical issues and the diversity of designers, this tool may soon become obsolete [Myers et al. 2000]. So the capability of integrating diverse tools may be more important than the functionality of the tools themselves.

4. SKETCHIFY: SKETCHING AS FLUENT EXPLORATION OF INTERACTIVE MATERIALS

In this section we present the basic idea and the principles behind Sketchify, an extensible toolset for sketching interactive systems.

4.1. Design Goals

The starting point for our work has been the ongoing discussion about the role of sketching in interaction design, especially the need to extend sketching from the creation of a pencil trace on paper towards dealing with other important attributes of the overall user experience, such as timing, phrasing, and feel [Buxton 2007]. We aimed at building a tool that can support designers in sketching novel interactive systems, and in doing so, we adhered to the following design principles.

—*Focus on supporting design of the overall user experience, especially the dynamics of the interaction*. Our main goal was to enable the designer to rapidly define sketches of interactive scenarios. This is expected to benefit both the designer and potential end-users, as they can *experience* the intended interaction at a much earlier stage in the design process, that is, well before extensive and detailed prototyping is attempted. We especially aimed at facilitating the definition of the dynamics and timing of such interactions.
—*Support the exploration of possibilities and limitations of relevant technologies*. We wanted to help designers to gain insight into new technologies through hands-on experience, so that they can develop more realistic expectations about the possibilities and limitations of the technologies they were considering. We chose to bring actual

samples of such technologies into the design space and to let designers use them as part of their sketches.

Having in mind the limitations of existing solutions, and in order to address the diversity of designers and the pace of technology change, we also kept some additional goals in mind.

—*Support more diversity and extensibility*. We aimed at providing a palette of alternative solutions from which designers can chose those elements that best match their skills and tasks. Meanwhile, we aimed at our solution being open in the sense that is should be relatively easy to add new tools and environments as they arise. This point of view was inspired by existing findings on how interaction designers actually use their tools. Stolterman et al. [2008] for example, described an interaction designer as a craftsperson,

> "someone who picks and chooses tools freely based on the situation and grounded in a judgment of overall benefits from using a specific tool. . . . 'benefits' have to do with so diverse aspects as the time available, the level of skill and mastery required, external pressure about standards, personal style of expression, etc." [Stolterman et al. 2008].

—*Provide orchestration and synergetic use of tools*. A crucial challenge when supporting sketching is how to provide coordination between the diverse tools that are available to stimulate the development of ideas. We kept the fluidity of sketch-based interaction in mind during development of the interaction with our own tool.

—*Better support for reuse of existing environments*. Instead of focusing on building yet another (specialized) sketching tool which could soon become outdated, we aimed at supporting designers in reusing their existing tools[11] and skills. Several design studies have indicated that interaction designers use rather different tools than the ones that HCI researchers are currently building [Stolterman et al. 2008; Stolterman 2008].

4.2. Conceptual Model

Sketchify implements our concept of fluent exploration of interactive materials by combining several existing and proven approaches, including freehand sketching, end-user programming, and I/O services. Figure 2 illustrates the conceptual framework of Sketchify. It distinguishes two groups of components.

—Tools that help a designer to create or bring into a design space interactive materials and services, for instance, through connections to simplified input/output (I/O) software and hardware services or links to external prototyping environments.

—Tools that enable a designer to sketch interaction by rapidly assembling these interactive materials, using freehand sketching, various forms of end-user programming, or paradigms already supported in available tools.

4.3. Integration and Variables

To integrate all elements of our solution, and to enable their orchestration and synergetic use, we applied a loosely coupled coordination model, where all elements of Sketchify communicate indirectly by exchanging messages through a centralized repository of variables. Alternatively, the tools can also communicate through files and

[11]To get a brief impression about the number of such available tools, visit www.dexodesign.com/2008/11/07/review-16-user-interface-prototyping-tools/, for a list of user interface prototyping tools, or *ACM Transactions on Graphics* Web site (tog.acm.org/resources/Software.html), for a comprehensive list of computer graphics software tools.

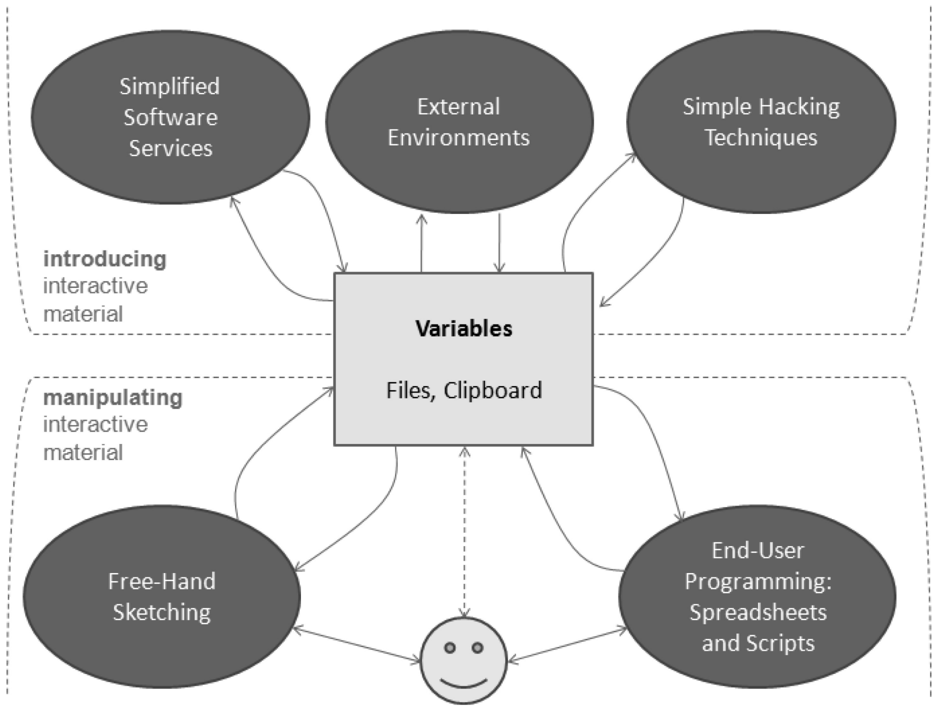

Simplified Software Services
External Environments
Simple Hacking Techniques
introducing interactive material
Variables
Files, Clipboard
manipulating interactive material
Free-Hand Sketching
End-User Programming: Spreadsheets and Scripts

Fig. 2. Sketchify combines free-hand sketching with support for state transitions and animation, end-user programming, and I/O services. Sketchify also offers the possibility to exchange information with existing software environments and simple hacking techniques. Blackboard architecture of globally available variables is used to connect all elements.

through the system clipboard. Sketchify runs a blackboard server with a simple repository of named slots, called variables, enabling external applications to update and read variables using one of the many available communication interfaces. We reused components from the Adaptable Multi-Interface COmmunicator (AMICO) project (see Appendix B for details) to implement this server. Variables provide a simple and uniform abstraction mechanism, enabling a designer to work with very diverse elements using the same set of operations. Properties of sketch elements, such as their position or transparency, or user actions such as item selections, can be mapped to blackboard variables. Spreadsheets or scripts can subsequently read, process, and update these variables. I/O services can receive arguments and send back results through such variables (Figure 3). Lastly, through extension mechanisms other platforms can update, read, or register for the notification of variables.

Our main motivations for using this abstraction were flexibility and simplicity. Untyped data structures, similar to our variables, have been widely used in other domains where heterogeneous applications need to work together [Edwards 2005]. In its basic ideas, the Sketchify middleware is similar to other loosely coupled and notification architectures, such as Elvin [Fitzpatrick 1999], and Lotus PlaceHolder [Dey 1999], as well as to tuplespace systems such as Linda [Gelernter 1985], iROS's EventHeap coordination layer [Johanson et al. 2002], and JavaSpaces [Freeman et al. 1999; Waldo 2000].

Sketchify variables are also easily manageable by end-users, as the concept of a single-address variables space is already familiar to many users through system variables and properties tables. This is confirmed by our initial studies with

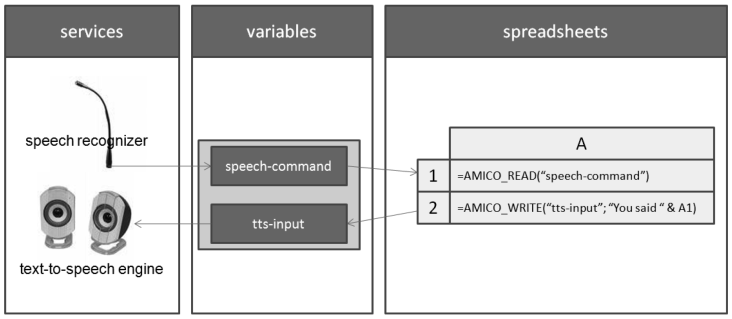


Fig. 3. An example of the communication among diverse elements of Sketchify through variables. In this example, a speech recognizer updates the variable "speech-command" with the recognized word. Within a spreadsheet, this variable is read, and the variable "tts-input" is updated in response. Update of this latter variable is propagated to a text-to-speech engine that pronounces the given text.

less-experienced users and students, which showed that such concept is indeed easy for them to understand. By simply reading, writing, and modifying the variables in a spreadsheet-like interface, users can, for example, directly try out and explore the basic functionality of interactive services without the need for advanced programming skills.

One of the main limitations of the Sketchify middleware is that variables can only contain textual data, and Sketchify modules cannot exchange images or binary objects through variables. However, it is possible to "hack" around these limitations by exchanging links to image files through variables, or by serializing graphical objects with textual or XML representation, as Sketchify can render an HTML and SVG encoded content.

In addition to exchanging messages through variables, tools can also communicate through files and through the system clipboard. For example, our freehand sketching environment saves graphical elements of sketches into image files that can subsequently be opened in most existing image manipulation programs. Through the system clipboard, images or text can be copied between our environment and many others.

5. INTRODUCING INTERACTIVE MATERIALS

We currently support three groups of tools that help designers to bring interactive materials into the design space, that is, I/O services, links to external development environments, and simple hacking techniques.

5.1. I/O Services

With I/O services, designers can introduce in their sketches real but "trimmed down" functionality of input or output devices and software components from various domains. We have incorporated many different services within Sketchify, including text-to-speech engines and speech recognizers, Web services (such as the Google spelling checker and search engine), Phidgets, Arduino, semantic services (such as the Wordnet definition service), camera-based face and motion detectors, MP3 and MIDI players, Wii Remote, and many others. Appendix D provides a complete list of the I/O services that are currently supported by our platform.

I/O services can bring within the reach of the designer a huge number of available software and hardware components. In order to accomplish this, we build on our previous work for integrating heterogeneous software components, where we used

Table I. Two I/O Services from the Domain of Speech Interaction, and the Variables They Use

I/O Service	Direction	Variables	Variable Description
FreeTTS text-to-speech engine	⇐	tts-input	Text to be pronounced.
	⇒	tts-status	Status of the engine: 'loading', 'ready', 'talking'
Sphinix-4 speech recognizer	⇒	speech-command	Recognized phrase
	⇒	sphinix4-status	Status of the engine: 'loading', 'ready', 'recognizing'

The designer sees and interacts with these services only through these variables, while Sketchify hides the complexity of the service execution.

a service-oriented approach[12] to connect components written in different languages [Obrenovic and Gasevic 2007]. In essence, our I/O services are stand-alone applications that Sketchify runs as background processes, and which connect to Sketchify through one of the many supported network interfaces, updating, and reading variables. Our services are simplified as we usually do not map the full functionality of the component, but only its most representative parts.

From the designer's point of view, Sketchify offers a simple interface to start and stop services, hiding the complexity and diversity of technologies that an I/O service may use, and providing a simple and uniform variable-based interface towards them (Table I). In this way, we can bring components from various domains within the reach of the designer, allowing a designer to directly experience possibilities and limitations of technologies without relying on programming skills.

5.2. Links to External Development Environments

Another way to bring into a design space examples of interaction modalities is to reuse components and examples from environments that are already used for the development of interactive systems. Many existing development and rapid prototyping platforms enable designers to define elements for user interaction. Such platforms come with lots of existing examples that can provide a good starting point for the exploration of novel interaction scenarios. In order to allow the designer to exploit the potential of these platforms, we currently support links towards several of them, including the following.

—Max, MSP and Jitter,[13] a high-level graphical environment for signal processing and music creation. The support of this platform for hardware devices and signal processing, for instance, is highly appreciated by our design students. This environment has been integrated using a Max/MSP Java extension mechanism that allows connecting Sketchify using Java network libraries. More specifically, we introduced two expressions, one for reading Sketchify variables, and another one for updating them.

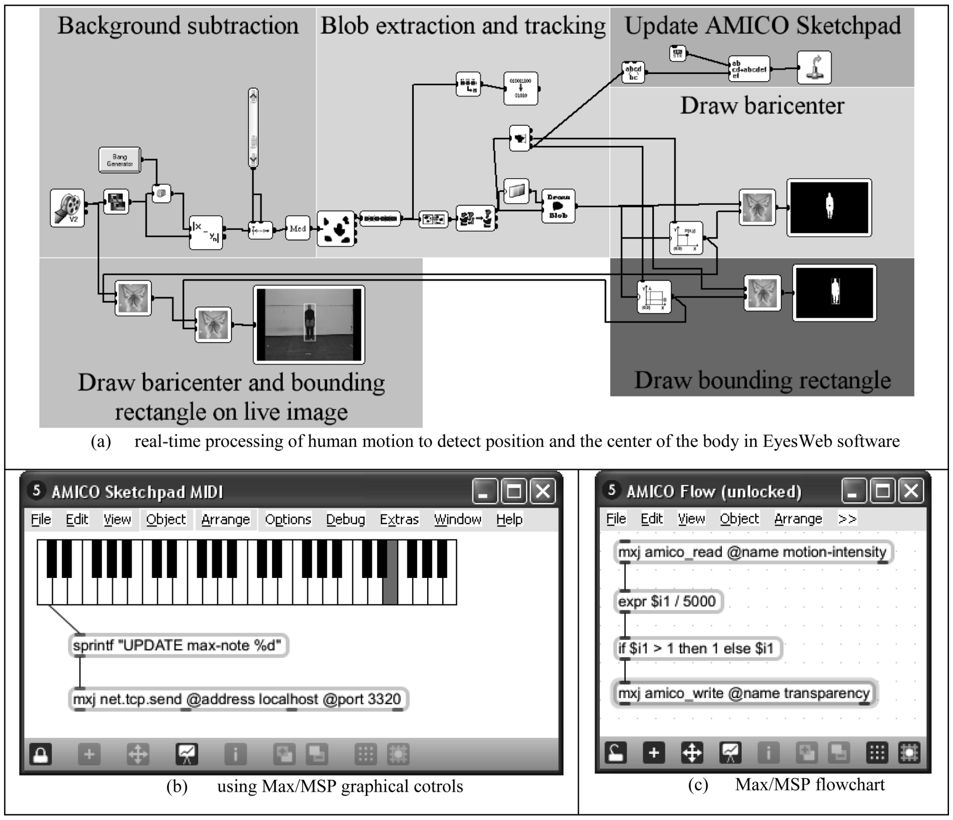
—EyesWeb[14] is a platform for interactive video processing. We integrated the environment using existing EyesWeb network components.

—Flash is an environment for the creation of interactive and animated (Web) applications. It is especially popular because of its powerful interactive graphics support. We created a simple library that builds on Flash XML/TCP support in order to connect Flash applications to Sketchify.

[12]We define a software service as a self-contained functional unit in which service consumers interact with the service through a well-defined interface. In this model, the consumer does not know (or care) "how" the service implements the requested action, only that the service performs "what" is defined by its published interface.

[13]http://www.cycling74.com/

[14]http://www.infomus.dist.unige.it/EywIndex.html


Background subtraction
Blob extraction and tracking
Update AMICO Sketchpad
Draw baricenter
Draw baricenter and bounding rectangle on live image
Draw bounding rectangle
AMICO Sketchpad MIDI
File Edit View Object Arrange Options Debug Extras Window Help
AMICO Flow (unlocked)
File Edit View Object Arrange >>

(a) real-time processing of human motion to detect position and the center of the body in EyesWeb software

```
sprintf "UPDATE max-note %d"
mxj net.tcp.send @address localhost @port 3320
```

(b) using Max/MSP graphical cotrols

```
mxj amico_read @name motion-intensity
expr $i1 / 5000
if $i1 > 1 then 1 else $i1
mxj amico_write @name transparency
```

(c) Max/MSP flowchart

Fig. 4. Connecting the EyesWeb (a) and Max/MSP (b) and (c) environments with Sketchify.

—Programming languages such as Processing, Java, C++, and C# are also supported. In these cases we created simple libraries on top of existing support of these languages for TCP and UDP protocols.

In all cases, we have extended the environments with remote access to the Sketchify variables, enabling external applications to interactively read and update these variables. For example, Figure 4(a) shows a screenshot of an EyesWeb example that processes human motion in real time, detecting the position and the center of gravity of a human body within the picture. Using an EyesWeb network sender component enables us to export the result of this processing by means of updates of Sketchify variables. In this way, we have been able to create sketches that illustrate how human motion can be used in interaction, for example, to control a character in a game or to control the playback of music in an MP3 player. Figure 4(b) shows another example where an interactive control from the Max/MSP environment is used to update Sketchify variables. In this case, the MIDI keyboard control updates the variable "max-note" with a number representing the note being pressed. The available links between the development platforms and our blackboard of variables allow a designer to use the development paradigms supported by the former tools while manipulating elements of his solution. For example, Figure 4(c) shows how the Max/MSP flow-chart syntax can be used to define the behavior illustrated in Figure 1. In this example, we first read the Sketchify variable "motion-intensity," process it with Max/MSP expressions, and send the result of the processing back to Sketchify, updating the variable "transparency."

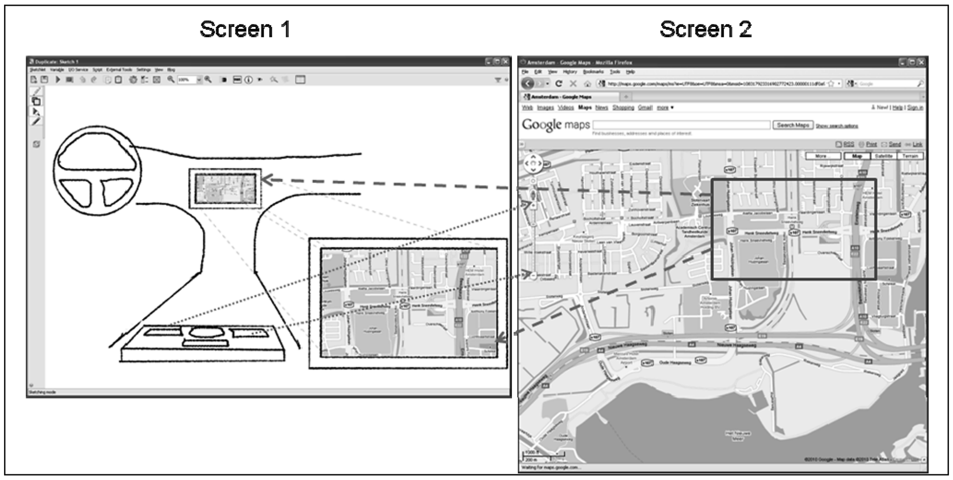


Fig. 5. Using screen capturing in combination with screen poking to illustrate a new interaction with a car navigation system. A part of the Sketchify sketch on the first screen dynamically captures part of the second screen, where a Web browser with the Google Maps application is running. Mouse clicks on the graphical regions in the sketch are mapped to mouse clicks in the application on the second screen, to zoom in and out and to move the map.

While our main goal has been to bring already existing functionality of development environments into the design space of Sketchify, designers could also use Sketchify as a supplement to such development environments, using Sketchify to add elements that these platforms do not support themselves, such as a text-to-speech engine or a speech recognizer.

5.3. Simple Hacking Techniques

To integrate pieces of software that do not have a programmable API and are not available as open-source components, we support some of the techniques used by hacking and mashup communities [Hartmann et al. 2008]. Sketchify includes several of such mechanisms, such as the following.

—*Screen scraping*, a technique based on parsing of a rendered user interface to gather data. In Sketchify, we enable designers to analyze and extract any part of a Web page in HTML or XML (such as RSS) formats.
—*Screen capturing*, a technique to dynamically bring any part of the screen as a part of the sketch, including output of video players. In Sketchify, such captured part can be manipulated and transformed as any other graphical object.
—*Screen poking*, a technique based on generating synthetic mouse and keyboard events computationally.

For example, Figure 5 illustrates using screen capturing in combination with screen poking, created by one of our students to illustrate a new interaction with a car navigation system. Two regions within the Sketchify sketch shown on the first screen dynamically capture part of the second screen, where an interactive map is shown in a Web browser. Mouse clicks on specific graphical regions in the sketch are mapped to mouse clicks for the application shown on the second screen. These mouse clicks are used to navigate through the map.

6. MANIPULATING INTERACTIVE MATERIALS THROUGH FREEHAND SKETCHING AND END-USER PROGRAMMING

As discussed, we provide several ways for manipulating interactive materials, including freehand sketching and various end-user programming paradigms. The main innovation in our support for freehand sketching and end-user programming is adding the link to Sketchify variables. Through this link, elements of freehand sketches, spreadsheet formulas, or script code can obtain access to various input and output devices, outside services, etc. On top of that, we also introduce specific support for controlling the dynamics of the interaction.

We first describe our freehand sketching environment, and then our support for end-user programming. Appendix A provides more details about the former, while Appendix C gives more details about the latter.

6.1. Freehand Sketching

With our support for freehand sketching, we want to exploit the freedom and expressiveness of pen-based gesturing. Our freehand sketching environment serves two functions. First, it facilitates the creation of the graphical elements that are part of an interactive solution. Second, it plays a key role in defining the dynamics of the interaction through sketch transitions, capturing of user events, and graphical transformations. Especially the latter aspect needs some explanation, as it is the most novel element of our solution.

From a designer's point of view, the environment looks very similar to a simple image editor, with additions for working with variables and with support for specialized tools, such as timers, which are useful when controlling the dynamic behavior of the interaction. Our environment supports standard options for freehand drawing, including setting stroke parameters, such as width and color. Our platform offers a limited number of colors and image manipulation options as this was considered to be sufficient in the conceptual stage of design. However, Sketchify can be configured so that, with a single mouse click, a user can open a sketch image in a more advanced image manipulation program, such as Adobe Photoshop. Next to the main drawing layer, we also provide an annotation layer, where designers can draw on top of the main sketch without affecting it. This latter feature is expected to be especially useful when discussing a sketch, with other designers or end-users.

Our freehand sketches consist of two types of elements: inactive elements (also called background images) and *active regions* (Figure 6). A background image is created by means of pen strokes or can be imported from an image file. An *active region* is a rectangular part of the sketch that can contain drawings, text, or images. Active regions can capture mouse (or pen) events and can update variables in response to such events. They can also be graphically transformed (translated, rotated, skewed) in response to updates of variables.

6.1.1. Defining Interaction through Transitions among Sketches. One way of defining interaction within Sketchify is by creating transitions between sketches. Linking sketches and defining conditions for transitions between them is a key functionality of our system. In its simplest form, Sketchify can define transitions in the same way as screen prototyping tools do, that is, in response to mouse clicks and keyboard events. Figure 7(a), 7(b), and 7(c) illustrate this by means of a simple example that consists of three sketches, each representing one screen of the interface.[15] The start screen (a) has two active regions that respond to mouse clicks. When a user clicks on the first (top) or second (bottom) active region a transition is initiated to sketch (b) or sketch (c), respectively.

[15]The example is based on the flip-book animation illustration from Buxton [2007, page 299].

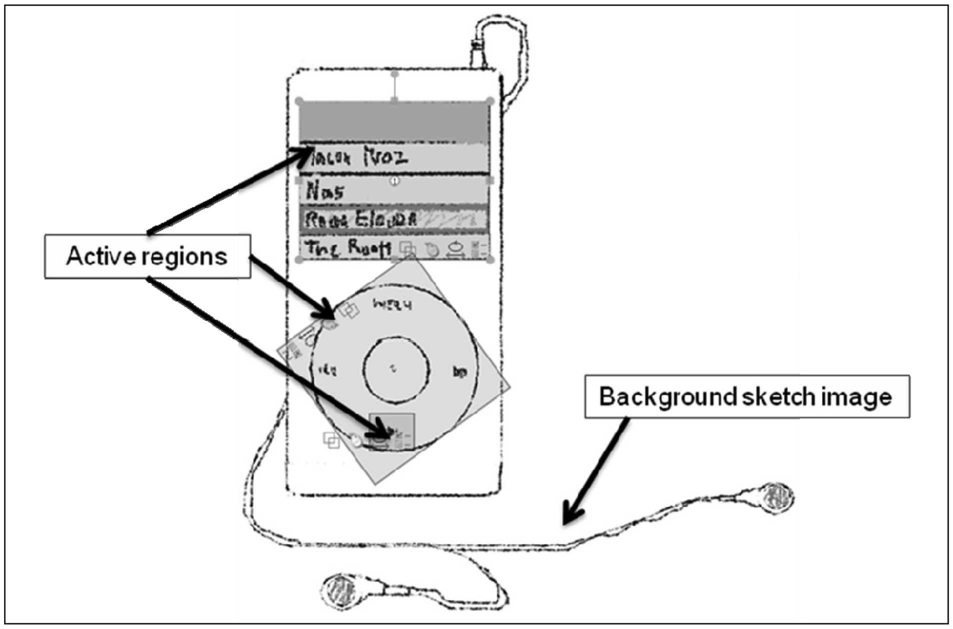


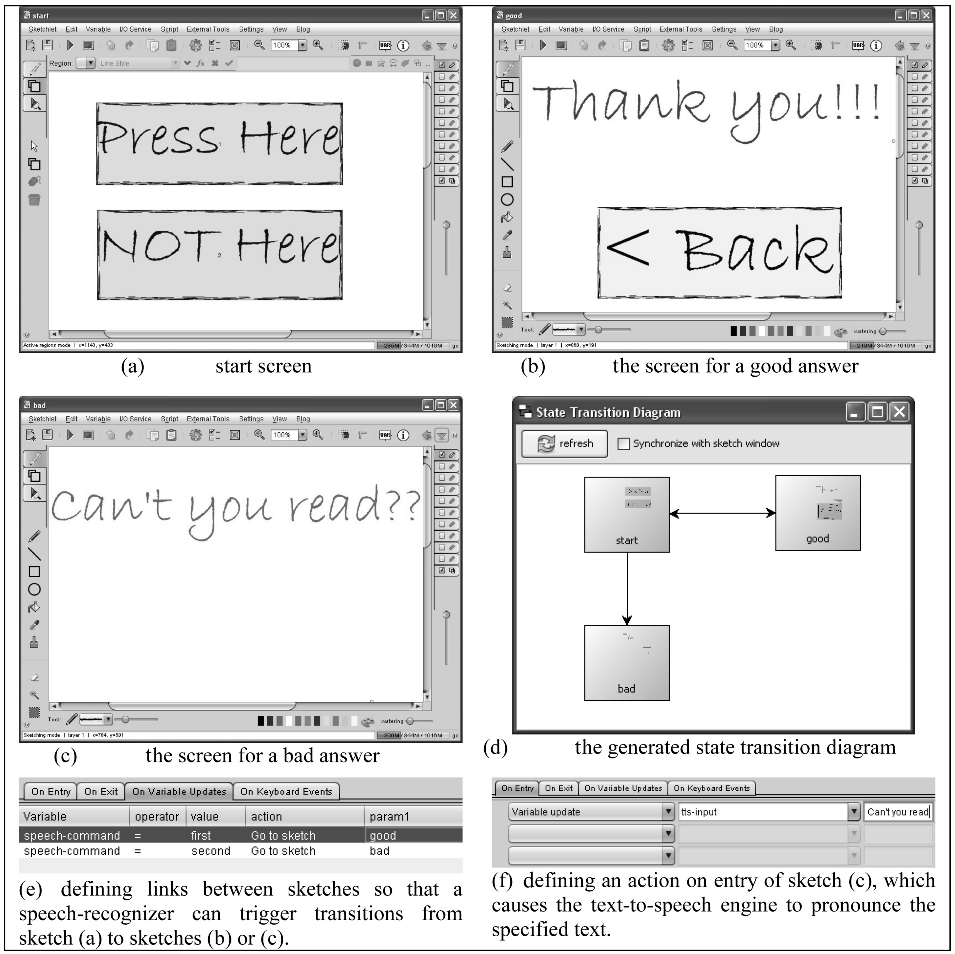
Fig. 6. An example of a freehand sketch created with Sketchify. In this example the sketch contains a background image, created with our freehand drawing tool, and three active regions that capture user mouse events and present additional graphics.

The second sketch (b) also has one active region, enabling the user to return to the start screen (a). The system automatically generates and displays a state transition diagram, which helps the designer to get an overview of the available sketches and the possible transitions between them (d). State transition diagrams are similar to the storyboards used in electronic sketching systems, but in our approach they are a side result of sketching, that is, they are created without requiring any explicit action on the part of the designer.

6.1.2. Presenting Variables through Active Regions. The support for state transitions, as described in the previous section, only allows the sketching of interaction at a relatively high level, where each interaction state is identified by an individual sketch. Through the combined use of active regions and variables, however, an individual sketch can also become “alive” and animated.

Active regions can be used to dynamically visualize data as we can control many properties of such active regions through variables, including their textual label, the path to the image file that they are associated with, or their geometrical properties such as position, orientation, and size. Figure 8(a) shows how the values of four numerical variables, which are updated in response to calculations that are performed within a spreadsheet, are converted into textual labels for active regions that are part of a freehand sketch. Figure 8(b) illustrates how the position of a face, as estimated by a face detector, can be mapped to the position of an active region.

Active regions can be constrained in terms of their maximum and minimum position and orientation. It is also possible to constrain the motion of an active region to a sketched trajectory. When such constraints are imposed, the position of the region can also be specified in relative terms, stating, for example, that the region should be positioned midway along the trajectory. An active region can also signal overlap with other regions and can trigger variable updates accordingly. In this way, dragging one



(a) start screen

(b) the screen for a good answer

(c) the screen for a bad answer

(d) the generated state transition diagram

Variable	operator	value	action	param1
speech-command	=	first	Go to sketch	good
speech-command	=	second	Go to sketch	bad

(e) defining links between sketches so that a speech-recognizer can trigger transitions from sketch (a) to sketches (b) or (c).

Variable update	tts-input	Can't you read

(f) defining an action on entry of sketch (c), which causes the text-to-speech engine to pronounce the specified text.

Fig. 7. Simple example that illustrates sketching with state transitions. The start screen (a) has two active regions that capture mouse clicks. When a user clicks on the first active region (top) a transition to the second sketch (b) initiates, while a click on the second region initiates a transition to sketch (c). The second sketch (b) also has one active region, enabling the user to return to the start screen (a). Transitions can also be triggered by variable updates (e), while a transition to a sketch can cause the update of a variable (f).

active region on top of another one may, for instance, trigger the transition to another sketch.

6.1.3. Updating Variables based on User-Triggered Events. Active regions can update variables in response to user actions, that is, they can react to characteristics of mouse motion (i.e., the distance, speed, and direction of movement) and mouse button events. By default, mouse dragging in combination with a left button press is mapped to the translation of an active region, while mouse dragging in combination with a right button press is mapped to the rotation of an active region.

Figure 9(a) illustrates this functionality by means of a simple interactive sketch of a children's audio book. On top of a background image, several transparent active regions are defined that capture mouse clicks, updating, in response, a variable "tts-input" with an a priori defined text. Update of this variable causes a text-to-speech engine to pronounce the given text. Figure 9(b) illustrates another example, where

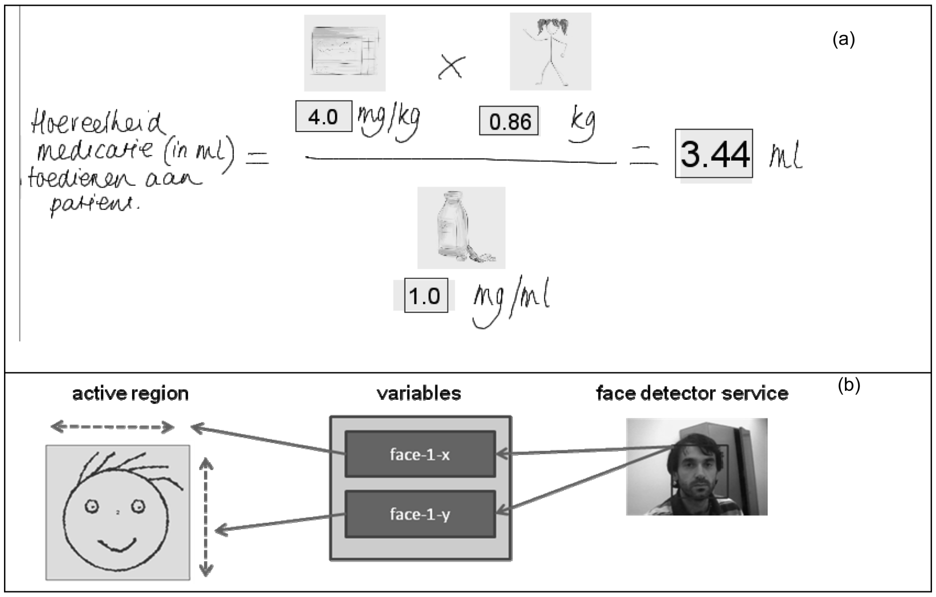


Fig. 8. Presenting variable values within sketches through updating the textual labels of active regions (a) or by translating an active region (b).

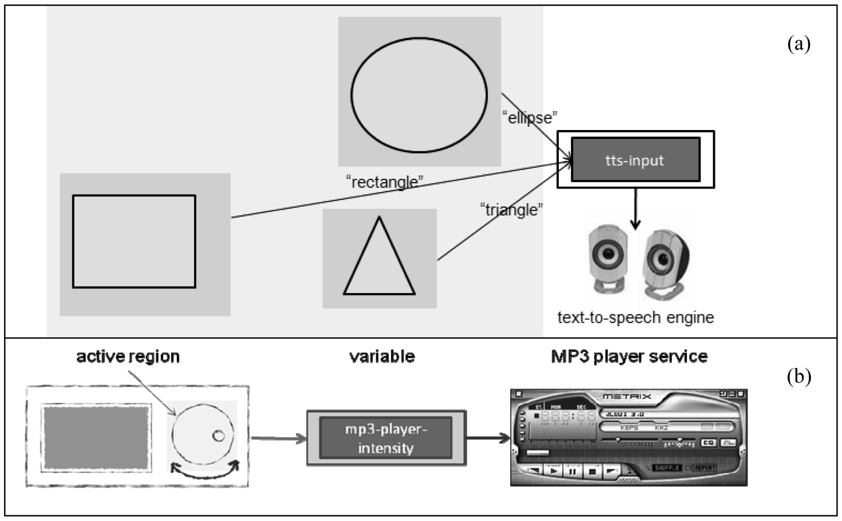


Fig. 9. Mapping mouse button clicks (a) and mouse movements (b) to variable updates that trigger interactive events, in this case output of the text-to-speech service, and changing the volume of the MP3 player.

the orientation of an active region is mapped to the volume control of an MP3 player, enabling a user to control the volume by rotating the active region.

6.1.4. Sketching as a Visual Coordination Language. Active regions can be connected to more than one variable, and the update of any of the connected variables will result in an update of all associated variables. Therefore, active regions can be used to connect variables in a simple and intuitive way. Figure 10 shows how this type of mapping can

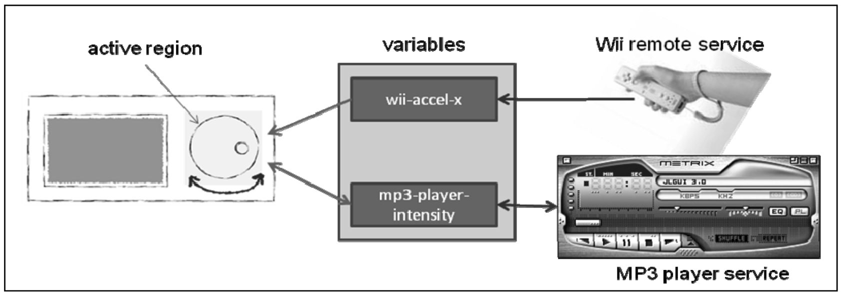


Fig. 10. Using graphical transformations as a visual coordination language: mapping the output of a Wii remote accelerator to the rotation of an active region in the sketch, and subsequently mapping this rotation to the volume control of an MP3 player.

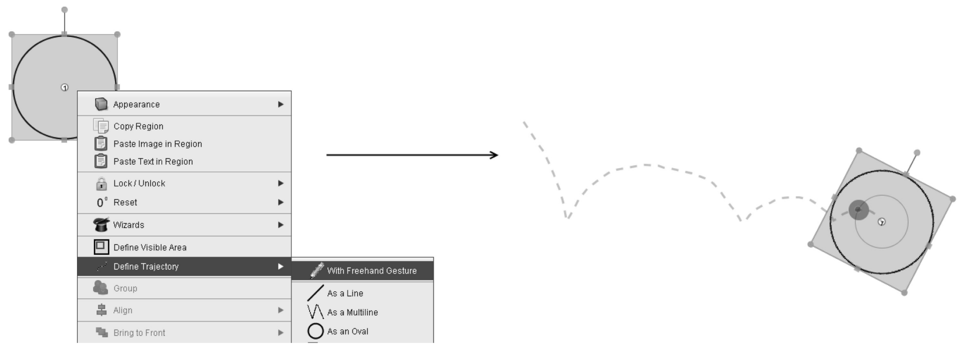


Fig. 11. Using a gesture to define the trajectory of an active region. Sketchify records the position as well as the velocity and acceleration of a gesture.

be used as a visual language to connect the motion detected by a Wii accelerator to the volume of an MP3 player.

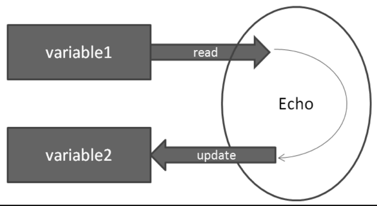
6.1.5. Defining Trajectories and Timers through Gestures. Sketchify exploits gesturing not only as a drawing modality, but also as a way to define a range of interactive effects. As stated before, the motion of a region can be restricted to a freehand sketched trajectory (Figure 11). Next to using gestures to defining trajectories, they can also be used to create timers with specific timer curves (see Appendix A for details). This means that such a timer can repeat the timing (velocity as a function of time) in the original gesture. In this aspect, Sketchify is inspired by the early work of Ronald Baecker and his GENESYS system for picture-driven animation [Baecker 1969a, 1969b].

6.2. End-User Programming

In order to define rich interactions, designers also need complex ways of influencing the behavior of sketches, for instance, by means of testing conditions, doing simple calculations, or creating sequences of actions. In many interaction scenarios, such as in speech applications, the sketches may not have any visible elements and the sketch behavior becomes the only "object" actually being designed. Having in mind that most designers are not experienced programmers and that there is a huge diversity between designers, we decided to connect our environment to a range of end-user programming tools which are likely to be accessible and usable for designers [Stolterman et al. 2008].

In Sketchify, spreadsheets and scripting languages can be used to quickly outline the behavior of sketches. Spreadsheets and scripts are proven, highly productive, and

Echo function



Spreadsheets implementation

	A
1	=SKETCHIFY_READ(“variable1”)
2	=SKETCHIFY_WRITE(“variable1-echo”; A1)

Javascript implementation

```
function variableUpdated(name, value) {
   sketchify.update( name + " -echo", “value is ” +
value);
}
```

Python implementation

```
def variableUpdated( name, value ):
   sketchify.update(name + “-echo”, "value is " +
value)
   return
```

BeanShell implementation

```
void variableUpdated( String name, String value,
String oldValue ) {
   sketchify.update(name+"echo","value is " +
value);
}
```

Ruby implementation

```
include Java

def variableUpdated( name, value )
  Java::Sketchify::Variable.update(name + '-echo',
name + ' = ' + value)
End
```

Prolog implementation

```
variable(Name,Value) :- sketchify _update("prolog-
echo",Value).
```

Sleep implementation

```
sub variableUpdated {
  $name = $1;  $value = $2;
   sketchify_update("$name-echo", "value is $value ");
}
```

Fig. 12. Examples of spreadsheets formulas and simple scripts written in different languages. All examples implement the same “echo” function: on update of one variable, another variable is updated with the same value.

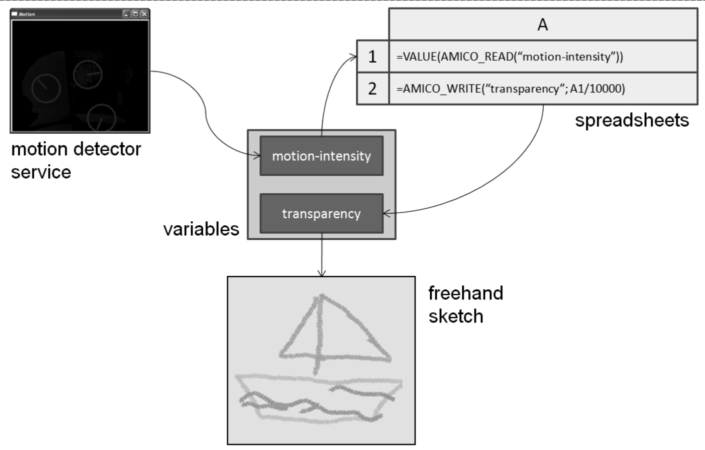
simple to learn and use end-user development paradigms [Obrenovic and Gasevic 2008, 2009]. We currently support OpenOffice.org CALC spreadsheets, and several higher-level scripting languages including Javascript, Python, BeanShell, Groovy, Ruby, TCL, Sleep, Haskell, and Prolog.

In all cases, existing end-user development solutions were extended with mechanisms to update and read Sketchify variables, to receive, cause, or process interaction events (Appendix C). Designers can use any of these individual end-user programming tools, or can combine them, describing, for example, a part of the behavior in spreadsheets, and another part in a script. Figure 12 illustrates how a simple “echo” behavior (on update of one variable, another variable is updated with the same value) can be accomplished with both spreadsheet formulas, and simple script code in six different scripting languages.

Figure 13(a) illustrates how spreadsheets and scripts can be used in combination with I/O services and freehand sketching in order to create a simple interactive sketch for the scenario as described in Section 1. The motion detector service tracks the intensity of the user’s motion in front of the camera, and updates the variable “motion-intensity.” This variable is passed to a spreadsheet that contains additional formulas to process this variable, more specifically, to map the motion intensity value into the range from 0.0 to 1.0. This derived value is passed into the new variable “transparency,” and the transparency of the freehand sketch responds to this variable. As a result, if the user is not moving, the image is invisible (completely transparent); the more she or he moves, the less transparent and more visible the image becomes. Figure 13(b) also shows how the same logic could be defined using a script instead of a spreadsheet.

Our support for end-user programming also allows for creating interactive sketches without graphical elements. For example, to sketch speech interaction, we may use a

(a) Interactive sketch with the spreadsheet logic



	A
1	=VALUE(AMICO_READ("motion-intensity"))
2	=AMICO_WRITE("transparency"; A1/10000)

(b) Javascript alternative for spreadsheet code

```
t = parseFloat(sketchify.get("motion-intensity")) / 5000;
sketchify.update("transparency", t > 1 ? "1" : "" + t);
```

Fig. 13. Implementation of the interaction sketch described in Section 1. The logic behind the sketch can be manipulated through spreadsheet formulas (a), or by means of a script (b).

spreadsheet containing only a speech recognizer and a text-to-speech engine. The example presented in Section 4.2.2 (Figure 3), for instance, illustrated such an interactive sketch.

7. CASE STUDIES

Sketchify has been employed in several educational activities at the Department of Industrial Design of the Eindhoven University of Technology. We have collected valuable feedback about benefits and limitations of Sketchify during these activities, which has helped us to obtain a more realistic impression of how Sketchify can be used most advantageously. We present three case studies describing usage of Sketchify in a group student project, in an individual student project, and in a course with 12 undergraduate students of industrial design.

7.1. Case Study 1: The "Snoet" Project

The first case study describes sketching with Sketchify by a group of three first-year students (in their second semester). We selected this case to illustrate how our platform can facilitate a sketch-like exploration of ideas through the combined usage of spreadsheets and I/O services, that is, even without involving freehand drawings.

The students were working on a product to help children in developing a healthy sleeping rhythm. They chose to use Sketchify in the last two weeks of their project, when they needed to come up with ideas on how to implement the imagined functionality of their system. They were learning to use Sketchify for the first time during this period. We observed several sessions in which they used Sketchify, we read their final project report, visited the exhibition were they presented their results, and asked them for additional details about their usage of Sketchify.

In the initial part of their project, following a literature review and some user studies, the students had identified the basic functionality that their product should have: the product should be able to detect if a child is awake, and then do some action to stimulate

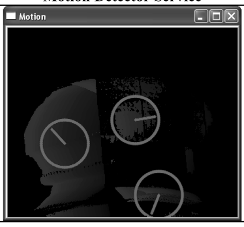
Motion Detector Service	Variables	Spreadsheet
	motion-intensity ⇨	=VALUE(SKETCHIFY_READ("motion-intensity")) =IF(A1>10000;"awake";"sleep")

Fig. 14. First "sketch" of a motion-based sleep-detector system.

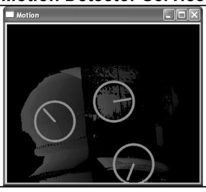
Motion Detector Service	Variables	Spreadsheet
	motion-intensity-1 motion-intensity-2 motion-intensity-3 … motion-intensity-10	=VALUE(SKETCHIFY_READ("motion-intensity-1")) =VALUE(SKETCHIFY_READ("motion-intensity-2")) … =VALUE(SKETCHIFY_READ("motion-intensity-10")) =AVERAGE(A1:A10) =STDEV(A1:A10)

Fig. 15. Slightly more advanced "sketch" of a motion-based sleep-detector system.

the child to fall asleep again, such as playing a song or playing a prerecorded message from the parents. The students, however, did not have any previous experience in working with sensing technologies, and they had only very basic knowledge about scripting languages and spreadsheets.

Before they came across Sketchify, their initial ideas included usage of EEG and biosignals, but they soon realized this to be unrealistic and too expensive. After being introduced to Sketchify in an informal meeting, they got interested in its usage as it could provide them with access to less expensive and more widely available camera-based detection techniques which they thought could be useful for their purposes.

Their first usage of Sketchify was to learn how a camera-based detector works. They used a motion-detector service available within Sketchify, and observed how the values changed when they moved objects or their body in front of the camera. They imported the value that represents the motion intensity into a spreadsheet, where they defined a simple threshold-based sleep detection mechanism. Figure 14 shows their first sketch. Even though it took them only ten minutes to create this solution, it has all the elements of their initial idea. Their design space consisted at that moment of two parameters, a "motion-intensity" variable which they manipulated by producing motion in front of the camera, and a threshold value which defined the transition between the "awake" and the "sleep" state based on the intensity of the motion.

After trying out this first sketch, they soon realized that the momentary motion value did not constitute robust information about sleep activity. They agreed that they needed an aggregate intensity of motion over some time period. The ability of our platform to serialize variable updates proved very useful, as they were able to derive several variables based on sequential updates of the motion intensity variable. As they were not sure what kind of processing was necessary, they started by importing serialized values of the motion detector into the spreadsheet. In the spreadsheet they experimented with several statistical functions that processed cells, starting with a simple averaging function. Figure 15 shows their new "sketch." Their design space now included additional parameters, such as a number of variables corresponding to restricted values of the motion detector and statistical functions that derive aggregate

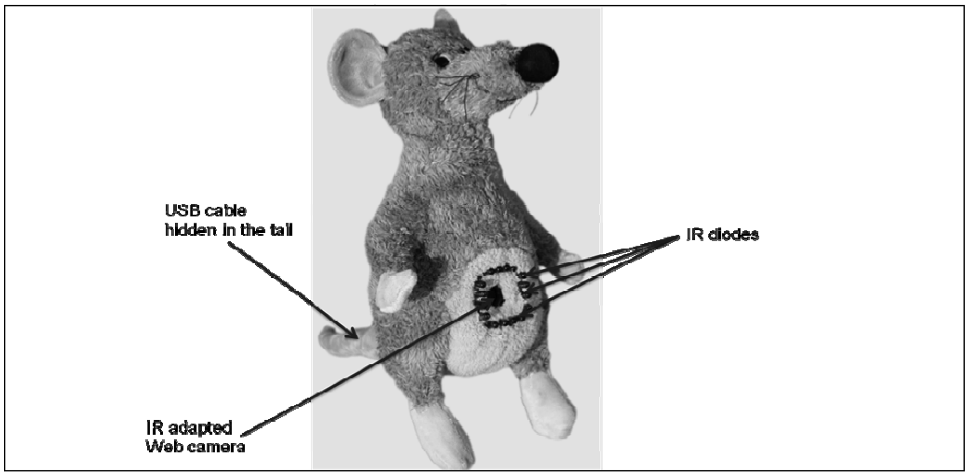


Fig. 16. Snoet, a toy that contains an IR-adapted Web camera and IR diodes that help the camera to "see in the dark." The toy is connected to the computer through a USB cable hidden in the tail of the toy.

values. Their exploration of this sketch revealed this to be a promising direction to pursue, as high-motion intensity over longer periods is more likely to reflect the "awake" state than a short period of intensity. They added a MIDI player service to play different notes in response to the threshold value being exceeded.

The students subsequently focused on how their solution could be made to work in dark conditions. They were, of course, aware that an ordinary Web camera is not adequate in such conditions. Following instructions found on the Internet, they turned the Web camera they had into a simple near-infrared (IR) camera. When they had the IR-adapted camera ready, they were able to try it with the interactive sketch they had built before, which revealed that it did not work very well.[16] The values received from the motion detector were very low and unreliable. After the initial disappointment, one of the students had the idea to use an IR light diode to shed more light on the area that was being monitored. They borrowed an IR diode from another group, and went into a dark room to test it, with much more encouraging results this time.

After this they spent the last week in adding more details to their solution, doing initial tests and preparing their final exhibition. They put the camera and IR diodes within the mouse toy (Figure 16), and made the processing more complex. They used a script to map motion intensity into 10 discrete values, created 160 serialized values, and calculated their standard deviation and average within a spreadsheet.

It is important to note that the students started their exploration without a clear understanding of the technology required, and that they did not have a clear idea about what they wanted to build. Nevertheless they managed to produce a functional system, and they received very positive comments during their final exhibition.

7.2. Case Study 2: The "Medical Assistant" Project

Our second case study describes usage of Sketchify by a third-year student of industrial design for her final bachelor project. The project focused on the development of an intelligent product to assist nurses in calculating the correct doses of medications. The student used Sketchify at two stages in the project: to sketch several alternative ideas

[16]The students did not realize that the adapted Web camera is a near-IR camera, not suitable for "night vision" as they expected.

and to discuss them with potential users, and to implement a prototype based on the selected concept. We selected this case to illustrate a joint usage of free hand sketches and spreadsheets, a frequently used combination of elements within Sketchify.

After doing initial studies and interviews, the student was familiar with the required medical calculations, and her task was to define an interface that would make working with these calculations more intuitive and more error resistant. Medical calculations were easy to implement within the spreadsheet extension of Sketchify, on top of which various freehand sketches were built and tested. Within these sketches the student explored various icons for the presentation of calculation parameters, as well as their spatial arrangement and transitions between them. Figure 17 shows examples of such interactive sketches. This example contains four freehand sketches with active regions that update and visualize variables imported and processed within the spreadsheet.

The student made a number of these projects with sketches, in total creating 20 variations, after which she selected four projects that looked most promising to her, and organized a small user study with six users. The study was organized as a talk aloud session, where the users could interact with the sketches and report which sketches they preferred and why. These outcomes informed the student about which design was the most successful one in making the content understandable, and she decided to further extend this selected design into a more advanced prototype.

The student also used our environment to implement her final prototype (Figure 18). Her reasons for doing this were that building the prototype with Sketchify was much easier and quicker for her, as she was able to implement all functionality within a spreadsheet, which was a very important criterion as she was not an experienced programmer.

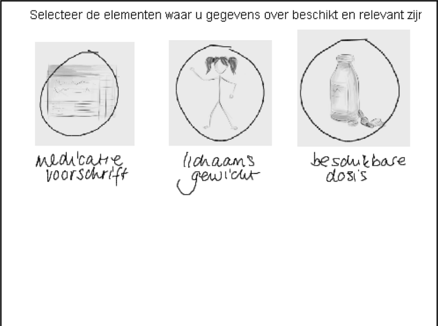
Parallel with her work on the software, she also explored how the interaction with the medical assistant could be made more tangible. For example, she used a prototype consisting of a touch screen covered by a transparent plastic plate with slider buttons. When the user presses or moves a plastic button, it is recognized as a mouse click or mouse move, and interpreted within the visual area.

The student created most of the freehand sketches with Sketchify using a graphical tablet, but she also imported some of the sketches that she initially had drawn on paper. The student's usage of a tablet input device also showed what hardware configuration could be necessary to support sketching with our platform. For example, the student spent most of her time in the laboratory where she was able to combine a tablet input device with digital pen and a primary screen controlled by keyboard and mouse. She used a tablet input device and digital pen to work with freehand sketches, but kept open a spreadsheet in the primary screen to work with formulas, modifying them using keyboard and mouse.

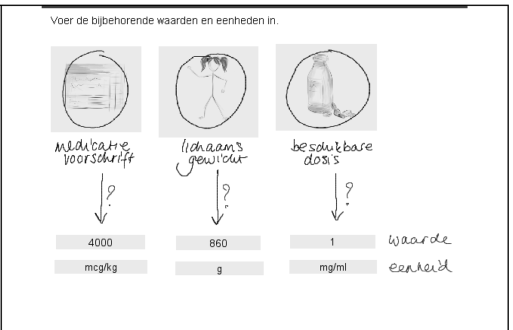
7.3. Case Study 3: Assignment "Sketching Interactive Systems"

With 12 undergraduate students (first to third year), we organized an assignment called "Sketching Interactive Systems".[17] The assignment lasted for seven weeks, where each week we organized meetings and discussions lasting 90 minutes, and students additionally spent one to two hours on individual work. The students were not restricted in terms of tasks they wanted to support. Rather we wanted to stimulate their creativity in using various sketching techniques to quickly build rough illustrations of their ideas in the area of novel interactive systems. The main objective of the course was to let students experience the design of interactive systems that use various novel interaction modalities, such as speech- and camera-based sensors, but also input from

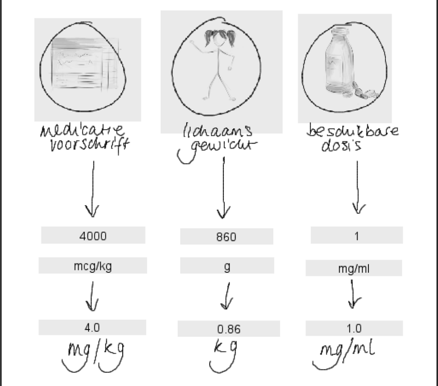
[17] http://www.vip.id.tue.nl/teaching/dg230_2008/


Selecteer de elementen waar u gegevens over beschikt en relevant zijr
medicatie voorschrift
lichaams gewicht
beschikbare dosis

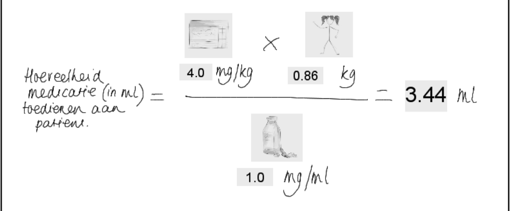
(a) selection of input variables to use within calculations



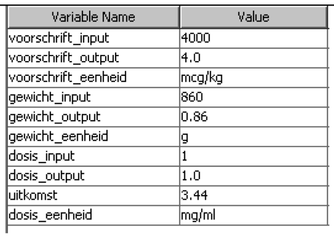
(b) data input (specifying values and units) for the selected input variables



(c) transformation of input variables to other units

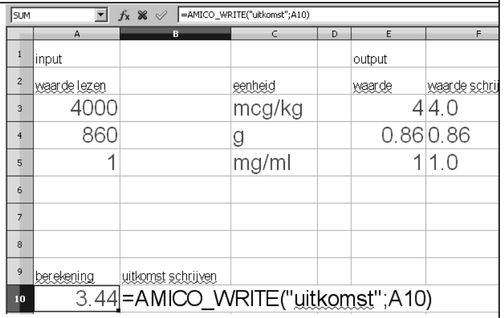


(d) visualization of the result



Variable Name	Value
voorschrift_input	4000
voorschrift_output	4.0
voorschrift_eenheid	mcg/kg
gewicht_input	860
gewicht_output	0.86
gewicht_eenheid	g
dosis_input	1
dosis_output	1.0
uitkomst	3.44
dosis_eenheid	mg/ml

(e) variables used to connect drawings with spreadsheets



SUM =AMICO_WRITE("uitkomst";A10)

	A	B	C	D	E	F
1	input				output	
2	waarde lezen		eenheid		waarde	waarde schrij
3	4000		mcg/kg		4	4.0
4	860		g		0.86	0.86
5	1		mg/ml		1	1.0
6						
7						
8						
9	berekening	uitkomst schrijven				
10	3.44	=AMICO_WRITE("uitkomst";A10)				

(f) spreadsheets formulas

Fig. 17. Sketches from the "Medical Assistant" project. On a first screen (a) the user selects which elements he or she wants to specify. On a second screen (b) values and units are specified. These values are stored in the variables that are imported in the spreadsheet (e). The third screen (c) shows these values, as well as values calculated within the spreadsheets in order to transform user input into alternative units. The last screen (d) visualizes these values as a formula, and presents the amount of medication required as determined from the calculations in the spreadsheet.

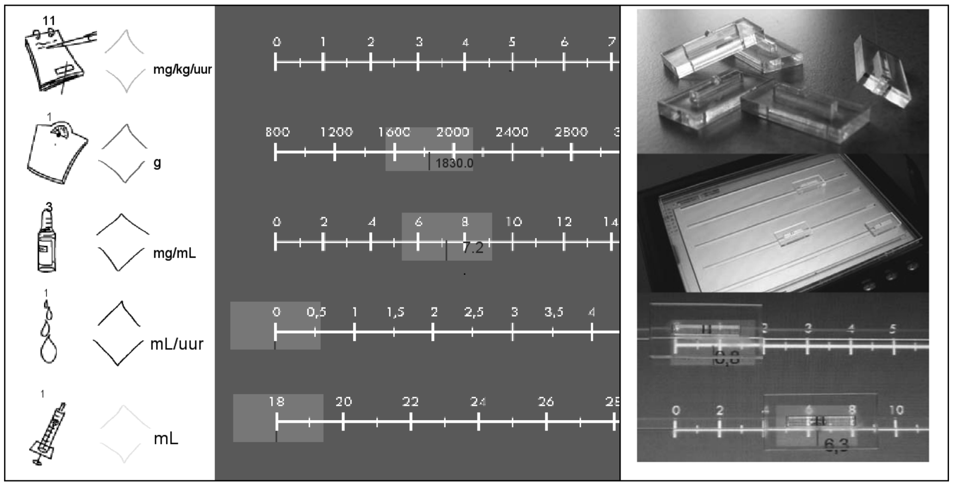


Fig. 18. User interface of the "Medical Assistant" prototype (left). A transparent plastic layer was used on top of a touch screen to add tangibility to the prototype (right).

Web services and other applications. The students did all assignments on their laptop computers.

We asked students to keep a creative log book in which to write down what they had learned and to reflect on the techniques they were using. These student logs provided us with in-depth feedback about how they experienced Sketchify. During weekly meetings we also promoted sketching as a teaching method. Although we prepared lots of material, during the meetings we were sketching "live," taking student discussions into account. This made our meeting more interactive, and during these meetings the students also came up with novel ideas.

In general we received very positive feedback from students. All students, including the first-year students, managed to incorporate novel interaction modalities in their projects, and made working interactive sketches. Students produced more than 200 projects with interactive sketches (average 13.4 projects per student, ranging from 6 to 30 projects per student). All students said that they planned to use Sketchify in their future projects, and some of them used it in their ongoing project, outside the class. Figures 19 and 20 show some of the produced interactive sketches.

In several of their projects students combined I/O services, end-user programming, and auxiliary tools, without using freehand sketches. For example, with our keyboard and mouse simulator, one student explored how face motion and speech can be used to control a range of applications. Moving the head left or right, for example, is an intuitive way to control walking of a "drunk" character in and online game (Figure 21(a)). In a more extended version, the same student also tried out how vertical head motion could be used in combination with speech to control the iTunes application (Figure 21(b)). Head motions are used to scroll the list of songs, and speech to select a genre or to control the playback.

7.4. Lessons Learned

We received very positive feedback from students and designers. The access to interactive environments and services has been shown to provide useful support for creating the dynamics and "feel" of interactive user interfaces. Students mostly experienced problems when using scripting, or when several components of our system were used simultaneously.

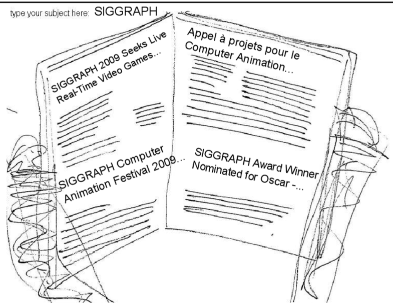
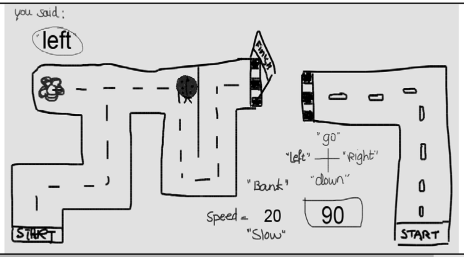
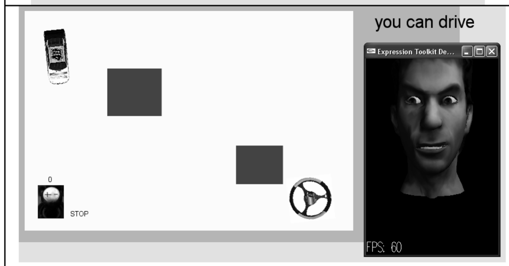
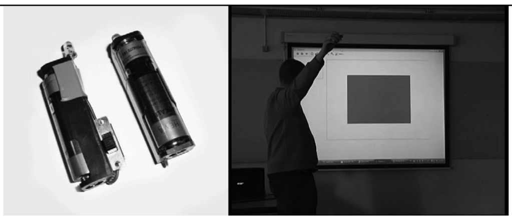
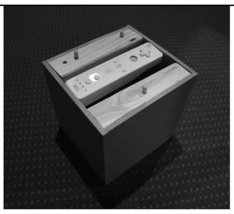
 type your subject here: SIGGRAPH SIGGRAPH 2009 Seeks Live Real-Time Video Games... Appel à projets pour le Computer Animation... SIGGRAPH Computer Animation Festival 2009... SIGGRAPH Award Winner Nominated for Oscar -...	**Google Newspaper** This sketch illustrates how the output of the Google news Web service can be presented in a form that visually resembles a real newspaper. A user can enter keywords, upon which the sketch refreshes its content through active regions that present the result of the Web service search. Used I/O services: Google news Web service
 you said: left "go" "Left" "Right" "down" "Bank" Speed = 20 "Slow" 90 START START	**Lady Bug** In this sketch, a user uses speech to control the motion of an animated character. In the background, spreadsheets are used to detect various conditions, and to control the speed of the bug motion Used I/O services: Speech recognizer Used EUP tools: spreadsheets
 you can drive Expression Toolkit De... 0 STOP FPS: 60	**Driving Instructor: Machine Emotions** In this sketch, a user can navigate a car on the screen by means of a mouse or Wii remote. A script tracks how well the driver is avoiding obstacles, and turns the number of errors into events that control the emotions in the face expression service. Used I/O services: Face expressions, Wii remote Used EUP tools: BeanShell scripts

Fig. 19. Some of the early interactive sketches created by students.


(a)


(b)

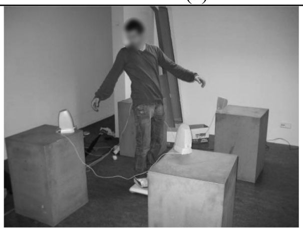
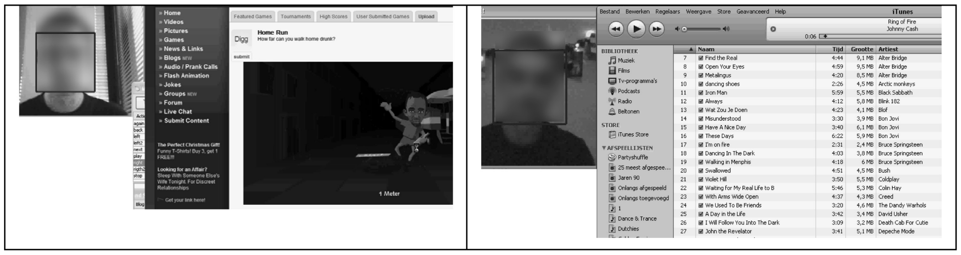

(c)

Fig. 20. Various projects using Wii devices. (a) a student exploring two-hand interaction using a Wii IR sensor and two IR diodes; (b) a Wii remote in a wooden box which could detect six discrete states depending on the rotation of the box; (c) a student controlling sound in space using WiiFit.



▲	Naam	Tijd	Grootte	Artiest
7	Find the Real	4:44	9,1 MB	Alter Bridge
8	Open Your Eyes	4:59	9,5 MB	Alter Bridge
9	Metalingus	4:20	8,5 MB	Alter Bridge
10	dancing shoes	2:26	4,5 MB	Arctic monkeys
11	Iron Man	5:59	5,5 MB	Black Sabbath
12	Always	4:12	5,8 MB	Blink 182
13	Wat Zou Je Doen	4:23	4,1 MB	Blof
14	Misunderstood	3:30	3,9 MB	Bon Jovi
15	Have A Nice Day	3:40	6,1 MB	Bon Jovi
16	These Days	6:22	5,9 MB	Bon Jovi
17	I'm on fire	2:31	2,4 MB	Bruce Springsteen
18	Dancing In The Dark	4:03	3,8 MB	Bruce Springsteen
19	Walking in Menphis	4:18	6 MB	Bruce Springsteen
20	Swallowed	4:51	4,5 MB	Bush
21	Violet Hill	3:50	5,5 MB	Coldplay
22	Waiting for My Real Life to B	5:46	5,3 MB	Colin Hay
23	With Arms Wide Open	4:37	4,3 MB	Creed
24	We Used To Be Friends	3:20	4,6 MB	The Dandy Warhols
25	A Day in the Life	3:42	3,4 MB	David Usher
26	I Will Follow You Into The Dark	3:09	3,2 MB	Death Cab For Cutie
27	John the Revelator	3:41	5,1 MB	Depeche Mode

Fig. 21. Using a face detector and a speech recognizer in combination with screen poking to control applications. Horizontal face coordinates control the online game (left), while vertical face movements and speech control the iTunes software (right).

7.4.1. What Worked Well. Our support for freehand sketching, spreadsheets, I/O services, and variables, in various combinations, proved to be easy to use and understand for all the students.

—*Freehand Sketching and State Transitions.* Almost all students immediately understood how to create freehand sketches, and created simple interactive sketches by using active regions and state transitions "state transitions . . . are the most basic tools that allow for simple interactive sketches . . . they have been useful and pretty straightforward" [S5]. Our support for sketching interaction through state transitions did not require lots of learning and work in addition to creating the drawings "[they] can make a sketch instantly interactive" [S11] "can help you communicate your ideas . . . without too much side-stuff" [S1].

Students also appreciated the state transition diagram that was automatically generated by our tool "by using state transition diagrams it is possible to get an overview of the system" [S4], and its value in understanding the logic of the interface "in a simple diagram with boxes and arrows, like a state transition diagram, you can see in a few seconds the logic thought behind the interaction" [S6]. An additional benefit of such diagrams is their ability to show the complexity and to identify missing links among states of the interface "With the state transition diagram I could clearly see which sketch was linked to another sketch, the diagram also clearly showed where links were missing or [which] sketches had too many links. It surprised me how fast the system state diagram could get complicated." [S3]

Students also pointed out the need for state diagrams at the level of a single sketch. That is, our state transition diagram treats a sketch as a one discrete state, but when variables and scripts are used, a sketch itself can also have quite a complex state transition diagram, which we currently cannot visualize.

—*Spreadsheets.* Not surprisingly, spreadsheets worked well for most users. We think that one of the reasons is that spreadsheets are similar to freehand sketches is two aspects. Firstly, they allow direct viewing and manipulating of data. Changes in formulas and cells have an immediate and direct effect on all dependent data. Second, they assist human spatial perception and reasoning: spreadsheets are designed to perform general computation tasks using spatial relationships rather than time as the primary organizing principle. Many people find it easier to perform calculations in spreadsheets than to write the equivalent sequential program [Chang 1990; Shu 1989; Nardi 1993]. The ability to define a set of cells with a spatial relationship to one another, exploiting users' natural spatial perception and reasoning, is one of the key properties underlying the success and widespread use of spreadsheets.

The ability to create more complex logic in sketches by including spreadsheets and scripts was positively appreciated by students. Spreadsheets and scripts allow students to express and develop their ideas more deeply and elaborately "it gives the sketch a whole new impulse" [S7], "add[s] a level of intelligence to sketches which is impossible to do without them . . . [such as] complex calculations, advanced comparisons, random number generation" [S4], "sketches can be more complex and it can give users more options and freedom" [S3], "the user will experience a smarter system that's able to make calculations and compare values, because of this I can communicate more complex ideas" [S5].

Such elements can also help you to think about important elements of your idea, "forces you to think about the logic behind your sketch. This can already give an initial discussion point" [S10]

—*I/O Services.* I/O services were very well received by students. In general, students acknowledged that I/O services have made their sketches more alive and brought them closer to the domain they were addressing "[they] can make the interaction with sketches richer and realistic" [S8], "make a sketch more alive" [S11], "the use of developed or partly developed software or devices in enhancing the overall experience of a sketch/model" [S2], "opens a world of new possibilities for more complex behavior of my sketches and it adds an extra level of realism. I/O services provide a glimpse of how it could be to integrate complex software input and output devices into my design and they're quick and easy to use" [S4] "In combination with the I/O services the possibilities for interactive sketches are limitless" [S10].

Most of the students encountered the technology used in our I/O services for the first time. The I/O services helped to deepen their knowledge of interaction technologies "offers possibilities which I didn't know they existed" [S4], "they broaden the possibility of sketching interactive systems" [S5] "enables me to give another approach of interaction with a device rather than just pressing buttons" [S8]. I/O services also help to raise more realistic expectations about sensing and intelligent technologies "good to practice with and to become aware that there are many possibilities . . . I [also] became more aware of the complexity of [such] software" [S6]

Students often came with new ideas after being inspired by existing I/O services. "I created scenarios so when I was creating them I was inspired by the different I/O services that are currently available" [S10] "It wasn't really that I started with an idea and then used the I/O service. It was more creating an idea [that] could have the I/O service in the sketch. But the I/O services were very easy to use, through the variables which could be called and changed" [S11]

Regardless of their practical value, one of the students noted that "I/O services were fun to use" [S6].

—*Variables and the Blackboard: More abstract Approach to Sketching.* One of the surprising pieces of feedback of students was their positive acknowledgement of variables and our interface to them. For us variables are background objects that allow connecting of components. What we did not realize is that many students never had had the opportunity to actually perceive live data from sensors or services. With variables, they could not only perceive them but could also play with them, for example, by simply updating a value in the spreadsheet interface, they could make the results of a service visible. Seeing how variables changed also provided indirect cues, such as the frequency of updates or the amount of noise in the sensor data.

The level of abstraction introduced by variables also enabled students to think in more abstract and general terms about interaction. Variables helped students to look at human-computer interaction in a more abstract way. As variables provide a uniform abstraction of very diverse elements, the students could now think about significantly different components in identical terms (variables), and consequently realized that there are few limits to what can be connected "In a more general sense I am now aware that almost every sensor can easily be connected with the PC and controlled, for example, using a Phidget set or a library for the Nintendo Wii controller." [S10] "There are a lot of events, like clicking with mouse, rotating with mouse, or using the keyboard. Later in the assignment I also learned about using external devices as triggers, like the Wii controller or a face detector." [S6].

7.4.2. What Did Not Work Well. Students did experience several problems when using scripting, or when several components of our system were used simultaneously.

Most of the problems that students reported were related to the usage of scripting, and almost all of them reported some problems and difficulties when working with scripts.

We identified two main sources of such problems. The first problem was relatively poor debugging support on our platform, as messages related to script errors were often not very informative. This remains an open problem, as for this feature we are relying on external libraries. The second, and possibly bigger, problem was that scripting requires a completely different style of interaction compared to freehand drawing and spreadsheets. In the latter activities, there is more freedom and visibility, and every change causes immediate and visible effects, which makes identifying errors relatively easy. While in spreadsheets you can immediately see and manipulate data, scripting is a much more indirect way of controlling behavior. The user first creates code, saves and reloads it, which are all processes with potential errors due to the strict syntax that is imposed, and then tests the script code by changing variables. This caused lots of confusions "when I changed something in the scripts, it didn't instantly work in the sketch. I had to refresh a lot of things, so that the scripting had effect on the sketch. So I wasn't sure whether the scripting was wrong, or whether the scripting hadn't been saved yet." [S11]. "With the scripting I didn't know if the script worked, because nothing changed" [S9]. The precision required by script code is also in contrast with the vagueness and ambiguity of freehand sketches "I found it hard to learn and very time consuming to program. The code has to be absolutely correct, missing a single dot can result in a program not working" [S4].

Some students found that scripting does not fit in the overall system "I think the strength of Sketchify is that it enables us to quickly sketch interactive systems without having to spend a lot of time on this. . . . However, scripting in Sketchify is quite difficult . . . so you need to spend a lot of time . . . this is a bit in contradiction with Sketchify's strength..." [S8].

An additional source of confusion was the difference between our variables and script variables. This caused problems for some of the students who tried to use Sketchify variables as if they were declared within the script. We were not able to solve this problem as we rely on third-party libraries for script support; however, we have recently added an easy copy-and-paste mechanism to the blackboard interface for creating expressions for accessing our variables from scripts.

Problems of Loosely Coupled Integration. Students also pointed out usability problems that they experienced when they were using the freehand sketching interface in combination with I/O services, spreadsheets, or scripts, as each of these elements are stand-alone applications, with their own interface. "Maybe less screens of everything. When I worked with sound services and scripts, I had a lot of windows open. It became difficult to find the right one." [S11]. One approach taken by some students working in our laboratory, such as our second case study shows, is to use two screens, one reserved for the freehand sketching environment, another for spreadsheets and I/O services.

Another problem is that external tools, such as image editors, are optimized for other tasks, and we can communicate parameters to them only in a limited way. For example, when students were using Adobe Photoshop to create and process images, they created images with print quality and a resolution that is much higher than what is required for a presentation within a sketch. When many such regions are used, this significantly reduces interaction performance without improving visual appearance. This problem may be partially resolved by subsampling the images when loading them into Sketchify.

7.4.3. Other Lessons. We summarize the other lessons learned into a number of categories: need for tools such as Sketchify, the value in improving the understanding and communication of interaction, the challenges of introducing complex ideas to users, and the way to cope with scripting limitations.

—*Need for Tools.* The benefits of Sketchify pointed out most frequently were its practical value in projects, improvement in the communication of dynamic effects and the feel of interfaces, a more in-depth understanding of interaction, and the ability to get early feedback from users and colleagues.

Our tool has been successfully used in several student projects, and students pointed out a need for such tools, especially when they had to work in novel domains, and with special user groups: "Since our users were elderly and the Web site was kind of confusing and it was hard to play around with the code to change things. This would have been way easier with Sketchify" [S1].

All students expressed their wish to use the tool in their future projects. "I definitely see me using Sketchify in the future because my vision is to design products that have an emphasis on rich playful interactions. I think that Sketchify is a good tool to make quick sketches without having to spend a lot of time on programming" [S8].

Students also see Sketchify as a good tool for getting early feedback from users. As Case Study 2 already illustrated, Sketchify enables designers to get early and more concrete feedback from their users before they make a serious commitment in the development of their idea. "The users have to use their imagination . . . which often results in distorted outcomes of early user tests. By using a tool like Sketchify it is really easy to make [a system] with which people can interact . . . . This way you can test different interaction styles and possibilities early . . . get reliable feedback on early sketches . . . " [S4]. In that way you could also check if you understood the user requirements correctly "you can ask for confirmation if this is the way [s]he meant it." [S10].

Students pointed out that this "also enables to provide others with better feedback" [S3], that is, "instead of saying 'I wonder what that would be like' I can actually say 'it works well' or 'it doesn't'" [S4].

—*Improved Understanding and Communication.* Students especially appreciated the ability of Sketchify to quickly create dynamic effects, through state transitions, animations, formulas, and I/O services, and its value in communicating their ideas about interaction. "I've seen absolutely beautiful sketches from people with excellent drawing skills, but as soon as complex movement or dynamics were involved, the sketches were a bit unclear. Extra explanation was needed for me to fully understand the ideas. When animated sketches are used, even people without excellent drawing skills could make their ideas clear to me: I could see in real time the movement and dynamics of the device, interface, or something else that is being sketched." [S4]

In this way, they were able to communicate in a more convincing way "showing the interactive sketch the product is more convincing for the people at the exhibition" [S6], and they were able to more convincingly demonstrate the behavior and feel of their system "by using external devices like the Wii, I am able to emulate/synthesize the feel of my sketch/model in a more interactive and natural way" [S2].

Seeing and experiencing interaction also makes it easier to understand its logic "If you see the system working it is easier to understand" [S3] "it is easy to understand interactions when I can experience [them] myself . . . experiencing interaction is always better than being told how it works" [S9] "How the idea should work/how you can interact with it is often vague. By using state transitions, [for example], the viewer has to use less imagination and the idea and interaction is more specifically visualized" [S10].

—*Introducing Sketchify.* Most users were able to quickly understand and adopt the idea of interactive sketching. However, these ideas needed to be appropriately introduced. Our platform itself does not impose serious limitations on ordering of actions, which may cause confusion and overload, as people could hardly grasp all possibilities that are available simply by looking at the controls, especially not when diverse elements were combined. We explicitly addressed these issues in two ways.

First, we created a number of introductory demonstrations. An introductory demonstration was usually sufficient to make the purpose of our tool understood, and a short training session enabled people to use Sketchify. A very useful way of introducing our tool was by means of video tutorials, which all students preferred to written descriptions. One negative feedback we received from students is the lack of such video tutorials for all options within the program.

Second, we adhered to the following multilayered design principles [Shneiderman 2003] when creating our tool, implying that functionality and options are gradually introduced to the users. Users typically start with the freehand sketching environment, where they learn how to use sketches to create and organize drawings. Subsequently, we introduced the concept of active regions and sketch transitions. After that, we introduced variables, and added operations that use variables as a means to dynamically influence freehand sketches. Once users were familiar with variables, we introduced spreadsheets and scripts. Last but not least, we showed how functionality could be added by means of I/O services.

—*Beyond Script Limitations.* Some of the students saw the strength of scripting not primarily in the ability to quickly create a piece of code from scratch, but rather in the ability to reuse already existing and tested code "I can use code I made earlier . . . or give it to someone else" [S6], "possible to implement code from external sources (made by others)" [S4]. One interesting idea for the future work is to incorporate script templates "script templates might make the scripting more [like] sketching" [S10], that is, creating a library of parameterized scripting functions that could be simply imported and easily changed.

Another approach we found in several projects is the combined use of very short scripts and spreadsheets, exploiting some complementary elements of these two paradigms. For example, students used spreadsheets for most calculations, and scripts for complex control structures such as nested IF statements. This is a somehow surprising finding, as it suggests that students were able to combine an assortment of tools in a way that a "real programmer" never would. While our initial idea of introducing multiple development paradigms was to enable students to *choose* among tools, this finding suggests that students also created a mental model where they where they *combine* programming paradigms in a new way.

8. DISCUSSION

As a summary of previous sections, here we discuss the benefits and limitations of our approach.

8.1. General Benefits of Our Approach

In comparison to existing solutions, our platform has several features potentially beneficial for sketching of interactive systems, including support for exploration of complex technologies in a simple way, diversity (both in terms of possible ways to interact with our platform and in the range of components being available) and reuse of existing environments.

8.1.1. Exploring the Possibilities and Limitations of Technologies. I/O services, although "trimmed down" versions of real components, bring "samples" of new technologies within the reach of the designer. By including such technologies in a sketch environment, we extended the design space, enabling designers to explore such services and to develop more realistic expectations about the possibilities and limitations of the technologies that these services rely on. For example, our students often came up with innovative ideas after being inspired by the possibilities offered by I/O services, reporting that they were not aware that such possibilities existed. On the other hand, designers can easily observe limitations of the technologies, such as the noise in the sensory data, the errors in recognition, or the delays in the response of Web services. This may stimulate them to find solutions that can help to overcome such limitations, or to make them more acceptable, in an early phase of their design.

8.1.2. Diversity of Components. We support many input and output devices, while in addition providing access to many external software components (see Appendix D for details), such as Web services and semantic services, which most other sketching or prototyping platforms currently do not support.

8.1.3. Extensibility and Domain Independence. We provide a number of extension mechanisms that can assist in adding even more external applications in the future. Diversity and extensibility make our platform less domain-dependent: any software component or service that can be mapped to variables can in principle be integrated into our toolset.

8.1.4. Reuse of Existing Environments. Our framework uses a range of existing environments. For example, our spreadsheet support is based on the OpenOffice.org CALC program, our scripting support reuses already existing contributions of the Java scripting project,[18] and while our freehand sketching environment has been built from scratch, it does allow a designer to open sketches in an alternative image editing program such as MS Paint or Adobe Photoshop. Through our I/O services and links to external environments, we aim to facilitate reuse of existing software and tools. This enables designers to reuse their skills and knowledge, which in turn is expected to lead to a faster and more efficient adoption of our toolset.

8.1.5. Diversity of Development Styles and Avoiding Proprietary Lock-In. With Sketchify, designers can be creative in selecting and combining their tools and development styles. Diversity in development styles is also an important requirement from the point of view of creativity support tools, where the following two basic principles have been identified as being important for their acceptance [Shneiderman 2007; Resnick et al. 2005; Myers et. al. 2000].

—*Low threshold, high ceiling, and wide walls.* In the words, make it easy for beginners to start (low threshold), but also enable experts to work on more complicated projects (high ceiling) and support a wide range of explorations (wide walls).
—*Support many paths and many styles*. That is, support adoption of different styles and approaches.

Most existing platforms provide one dominant development style. For example, electronic sketching tries to enable a user to express as much as possible through freehand

[18]https://scripting.dev.java.net/

gesturing, spreadsheets facilitate the creation of declarative relations among cells, Max/MSP allows dataflow specification, Flash supports scripting, and processing offers object-oriented programming. We instead enable a designer to choose among and combine diverse interaction paradigms. Interaction designers can select the paradigm closest to their skills or can combine paradigms, switching them when the limitations of one have been reached or are no longer appreciated.

Through its ability to work with diverse interaction paradigms and tools, our platform can help a designer to avoid the proprietary lock-in issue, that is, being too dependent on one vendor for products and services and not being able to move to another vendor without substantial switching cost. In our environment, the same task can often be realized with different tools, and designers can compare and test the limits of the used paradigms. For example, in their projects, students have often combined spreadsheets with scripting, where they have used scripting to overcome limitations of spreadsheets, especially when defining a control flow (for example, complex if-then-else scenarios).

An important side-effect is that our approach teaches students how to sketch in more universal terms that go beyond tools, enabling them to think about sketching as a way of developing ideas that can be implemented by a range of different tools, each of which, as technology develops, may be subject to substantial changes.

8.1.6. Promoting More Efficient Collaboration between Designers and Engineers. I/O services open a possibility for a more efficient interaction between designers and engineers. One of the problems that we have often experienced when designers and engineers need to work together is that the engineers perceive the ideas of designers as being unrealistic and not precise enough to be useful. Our I/O services, although simplified, resemble real components, and sketches expressed in terms of these services are more likely to be close to the implementation platforms that the engineers use. Through the exploration of services, the designer can develop more realistic expectations about the possibilities and limitations of technologies. This interaction between designers and engineers could work in two ways, where, in the early stages of design, engineers could provide designers with I/O services, adapting some of the components and services that they might use later on in the implementation stage. We provide lots of auxiliary tools that can assist engineers in this process [Obrenovic and Gasevic 2007]. This may also inspire a more general approach towards building software services and components, where each service could have two sets of Application Programming Interfaces (APIs), one engineering API with full functionality, and one sketching API representing a simplified and limited sample of the full functionality.

8.2. General Limitations of Our Solution

The approach taken by our platform also comes with some limitations. First, it is important to keep in mind that our platform is intended for sketching, and that, although we support a huge range of components and environments, these elements are simpler than equivalent elements in advanced prototyping and programming environments. We wanted to enable designers to quickly and roughly sketch interaction, rather than to create precise and high-fidelity prototypes. To simplify integration of existing software components, we compromised on issues such as performance or security, which are important engineering issues that cannot be ignored in the later stages of development. To summarize, our platform tries to improve diversity and freedom, which comes at the price of precision. This makes our platform unsuitable for the development of final products, and of limited use for the creation of real-time high-fidelity prototypes.

Loosely coupled integration of various environments, as we have adopted in our tool, can make usage of such tools tedious, especially when compared to a single integrated environment. This can results in many windows with a nonuniform look and feel being open at the same time. When many modules are used, the number of variables that a designer has to manipulate can also become significant, and finding the right variable may become difficult.

One of the benefits of our platform is extensibility, but adding elements, such as new I/O services or environments, requires involving people with some programming experience. We provide integration mechanisms that simplify this integration, but they, nevertheless, require some programming skills, and most designers will probably not be able to perform them by themselves. We partially remedy this problem by enabling nondevelopers to integrate existing environments through auxiliary "hacking" tools, such as a mouse and keyboard simulator or a screen scraping.

8.3. Implications for Developers of Design Tools: A Tool as a Service

We would like to encourage developers of new design tools to make their tools open and easy to integrate and combine with other tools, as the ease with which such tools can be integrated into existing environments can be equally (or sometimes even more) important than the key functionality of their tool itself. Many currently available design tools mainly support file-based interoperability, that is, the ability to import and export data in formats that are recognized by other programs. For supporting tasks such as sketching this is often not sufficient and a more synergetic and real-time integration of tools is required. In our experience, a service-oriented approach is a promising direction for more closely integrating diverse environments and components. Sketchify provides a demonstration that such an approach allows integrating a diversity of tools and components, irrespective of the fact that they might be written in very diverse languages and can rely on different technologies (see Appendix D for details).

Applying a service-oriented approach also simplifies writing extensions for tools such as Max/MSP or OpenOffice.org CALC, since many of these tools already provide mechanisms to extend their functionality. Rather than embedding the whole functionality of Sketchify within such a tool or vice versa, we have chosen to use a simple communication protocol based on reading and updating variables. Such protocols are easily supported by network connection mechanisms, one or more of which are usually available within current tools' extension libraries. Although such networking support has proven sufficient, it is currently mostly provided at a low level (i.e., in a form of a socket library).[19] We would like to encourage developers to also start supporting higher-level Internet protocols, such as XML-RPC, OSC, or HTTP, since this has the potential to make integration of tools simpler.

8.4. Future Work

In our future work, we plan to concentrate on facilitating design team work, and on supporting the collaboration of designers with relevant stakeholders, such as engineers, market experts, and end-users.

Our tool is currently conceived as a designer's personal sketchbook. We also plan to explore how it could potentially facilitate collaboration between designers, and be used to develop and document the work of a complete design team. One possible approach is to create a shared sketchbook, where designers (and engineers) in different roles can view, annotate, or change the sketches.

Sketchify has the potential to improve the communication between designers and other stakeholders in the business domain, as it enables a combination of tools that designers use (e.g., freehand sketching) with tools that people in the business domain use (e.g., spreadsheets). One possible approach is to create sketches and prototypes

[19]http://en.wikipedia.org/wiki/Internet_socket/

that can be adapted through spreadsheets, so that a design solution can be adapted to new situations by nondesigners. Another potential is using already existing business knowledge and logic, captured in spreadsheets, so that designers could build on top of such existing, and validated, material.

9. CONCLUSIONS

In this article we have described Sketchify, a tool for sketching interactive user interfaces. With Sketchify, we extended the concept of paper and pencil sketching towards the more generic concept of fluent exploration of interactive materials, enabling designers to create "interactive sketches" that illustrate interaction scenarios or interaction techniques. To stimulate further research in this direction, our software and other materials are freely available.[20]

We conclude by making two points. First, we support a view that sketching should be extended beyond the simple creation of a pencil trace on paper to deal with important attributes of the overall user experience, especially time, phrasing, and feel [Buxton 2007]. Second, some aspects of specifying interactive system behavior are beyond free-hand drawings and we need tools that can seamlessly integrate sketching with more traditional (end-user) programming techniques. Sketchify demonstrates that the combination of diverse environments can facilitate the development of ideas in a similar way as more integrated domain-specific sketching tools or paper and pencil sketching would do, at the same time offering many more possibilities.

ACKNOWLEDGMENT

We would like to thank B. Atasoy, J. Quevedo, B. Buxton, D. Frohlich, and L. Hardman for their comments. We also thank students of Industrial Design at the Eindhoven University of Technology for their feedback.

REFERENCES

AVRAHAMI, D. AND HUDSON, S. E. 2002. Forming interactivity: A tool for rapid prototyping of physical interactive products. In *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS'02).* ACM, New York, 141–146.

BAECKER, R. M. 1969a. Picture-Driven animation. In *Proceedings of the Spring Joint Computer AFIPS'69 Conference.* ACM, New York, 273–288.

BAECKER, R. M. 1969b. Interactive computer-mediated animation. Ph.D. thesis, MIT, MAC-TR-61.

BAILEY, B. P., KONSTAN, J. A., AND CARLIS, J. V. 2001. DEMAIS: Designing multimedia applications with interactive storyboards. In *Proceedings of the 9th ACM International Conference on Multimedia (MULTIMEDIA'01).* Vol. 9, ACM, New York, 241–250.

BUXTON, B. 2007. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann.

CHANG, S. K. 1990. *Principles on Visual Programming Systems.* Prentice Hall.

CHEN, X., KANG, S. B., XU, Y., DORSEY, J., AND SHUM, H. 2008. Sketching reality: Realistic interpretation of architectural designs. *ACM Trans. Graph. 27*, 2, 1–15.

DAVIS, J., AGRAWALA, M., CHUANG, E., POPOVIC, Z., AND SALESIN, D. 2003. A sketching interface for articulated figure animation. In *Proceedings of the ACM Siggraph/Eurographics Symposium on Computer Animation.* 320–328.

DAVIS, R. C., COLWELL, B., AND LANDAY, J. A. 2008. K-Sketch: A 'kinetic' sketch pad for novice animators. In *Proceedings of the 26th Annual ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'08).* ACM, New York, 413–422.

DEY, A. K., SALBER, D., ABOWD, G. D., AND FUTAKAWA, M. 1999. The conference assistant: Combining context-awareness with wearable computing. In *Proceedings of the 3rd IEEE International Symposium on Wearable Computers (ISWC)*. IEEE Computer Society, 21.

EDWARDS, W. K. 2005. Putting computing in context: An infrastructure to support extensible context-enhanced collaborative applications. *ACM Trans. Comput.-Hum. Interact. 12*, 4, 446–474.

[20]http://sketchify.sf.net/

FITZPATRICK, G., MANSFIELD, T., KAPLAN, S., ARNOLD, D., PHELPS, T., AND SEGALL, B. 1999. Augmenting the workaday world with Elvin. In *Proceedings of the 6th European Conference on Computer Supported Cooperative Work (ECSCW).* S. Bødker et al. Eds., Kluwer Academic Publishers, Norwell, MA, 431–450.

FREEMAN, E., HUPFER., S., AND ARNOLD, K. 1999. *JavaSpaces Principles, Patterns, and Practice*. Prentice Hall.

GELERNTER, D. 1985. Generative communication in Linda. *ACM Trans. Program. Lang. Syst. 7*, 1, 80–112.

GRADY, H. M. 2000. Web site design: A case study in usability testing using paper prototypes. In *Proceedings of the IEEE Professional Communication Society International Professional Communication Conference and Proceedings of the 18th Annual ACM International Conference on Computer Documentation: Technology and Teamwork.* 39–45.

GREENBERG, S. AND BOYLE, M. 2002. Customizable physical interfaces for interacting with conventional applications. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST'02)*. ACM, New York, 31–40.

GREENBERG, S. AND FITCHETT, C. 2001. Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST '01).* ACM, New York, 209–218.

HARTMANN, B., DOORLEY, S., AND KLEMMER, S. R. 2008. Hacking, mashing, gluing: Understanding opportunistic design. *IEEE Pervas. Comput. 7*, 3, 46–54.

HARTMANN, B., KLEMMER, S. R., BERNSTEIN, M., ABDULLA, L., BURR, B., ROBINSON-MOSHER, A., AND GEE, J. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and (UIST'06).* ACM, New York, 299–308.

IGARASHI, T. AND ZELEZNIK, B. 2007, Guest Editors' introduction: Sketch-Based interaction. *IEEE Comput. Graph. Appl. 27*, 1, 26–27.

IGARASHI, T. AND HUGHES, J. F. 2003. Smooth meshes for sketch-based freeform modeling. In *Proceedings of the Symposium on Interactive 3D Graphics (I3D'03).* ACM, New York, 139–142.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proceedings of the 26th Annual International Conference on Computer Graphics and Interactive Techniques*. ACM Press/Addison-Wesley, New York, 409–416.

JOHANSON, B., FOX, A., AND WINOGRAD, T. 2002. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervas. Comput. 1*, 2, 67–74.

KARA, L. B. AND SHIMADA, K. 2007. Sketch-Based 3D shape creation for industrial styling design. *IEEE Comput. Graph. Appl. 27*, 1, 60–71.

KELLEY, J. F. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Office Inf. Syst. 2*, 1, 26–41.

KLEMMER, S. R., NEWMAN, M. W., FARRELL, R., BILEZIKJIAN, M., AND LANDAY, J. A. 2001. The designers' outpost: A tangible interface for collaborative Web site. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST'01).* ACM, New York, 1–10.

KLEMMER, S. R., SINHA, A. K., CHEN, J., LANDAY, J. A., ABOOBAKER, N., AND WANG, A. 2000. Suede: A Wizard of Oz prototyping tool for speech user interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST'00).* ACM, New York, 1–10.

KRIPPENDORFF, K. 2006. *The Semantic Turn: A New Foundation for Design*. Taylor & Francis, Boca Raton, FL.

LANDAY, J. A. 1996. SILK: Sketching interfaces like krazy. In *Conference Companion on Human Factors in Computing Systems: Common Ground (CHI'96).* M. J. Tauber Ed., ACM, New York, 398–399.

LEE, J. C., AVRAHAMI, D., HUDSON, S. E., FORLIZZI, J., DIETZ, P. H., AND LEIGH, D. 2004. The calder toolkit: Wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS'04).* ACM, New York, 167–175.

LI, Y. AND LANDAY, J. A. 2008. Activity-Based prototyping of ubicomp applications for long-lived, everyday human activities. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems (CHI'08).* ACM, New York, 1303–1312.

LI, Y., HONG, J. I., AND LANDAY, J. A. 2004. Topiary: A tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST'04).* ACM, New York, 217–226.

LIN, J., NEWMAN, M. W., HONG, J. I., AND LANDAY, J. A. 2000. DENIM: Finding a tighter fit between tools and practice for Web site design. *In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'00)*. ACM, New York, 510–517.

MACINTYRE, B., GANDY, M., DOW, S., AND BOLTER, J. D. 2004. DART: A toolkit for rapid design exploration of augmented reality experiences. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04).* ACM, New York, 197–206.

MOGGRIDGE, B. 2007. *Designing Interactions*. MIT Press, Cambridge, MA.

MYERS, B., HUDSON, S. E., AND PAUSCH, R. 2000. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact. 7,* 1, 3–28.

NAM, T. AND LEE, W. 2003. Integrating hardware and software: Augmented reality based prototyping method for digital products. In *CHI'03 Extended Abstracts on Human Factors in Computing Systems.* ACM, New York, 956–957.

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

OBRENOVIC, Z. AND GASEVIC, D. 2007. Open source software: All you do is put it together. *IEEE Softw. 24,* 5, 86–95.

OBRENOVIC, Z. AND GASEVIC, D. 2008. End-User service computing: Spreadsheets as a service composition tool. *IEEE Trans. Serv. Comput. 1*, 4, 229–242.

OBRENOVIC, Z. AND GASEVIC, D. 2009. Mashing up oil and water: Combining heterogeneous services for diverse users. *IEEE Internet Comput. 13*, 6, 56–64.

OLOFSSON, E. AND SJÖLÉN, K. 2005. *Design Sketching*. KEEOS Design Books AB, also http://www.designsketching.com.

OWADA, S., NIELSEN, F., NAKAZAWA, K., AND IGARASHI, T. 2006. A sketching interface for modeling the internal structures of 3D shapes. In *ACM SIGGRAPH 2006 Courses*. ACM, New York, 12.

PERING, C. 2002. Interaction design prototyping of communicator devices: Towards meeting the hardware-software challenge. *Interact. 9*, 6, 36–46.

RESNICK, M., MYERS, B., NAKAKOJI, K., SHNEIDERMAN, B., PAUSCH, R., SELKER, T., AND EISENBERG, M. 2005. Design principles for tools to support creative thinking. In *Proceedings of the Workshop on Creativity Support Tools*. www.cs.umd.edu/hcil/CST/Papers/designprinciples.pdf.

RETTIG, M. 1994. Prototyping for tiny fingers. *Comm. ACM 37*, 4, 21–27.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A. 2005. ShapeShop: Sketch-Based solid modeling with blob trees. In *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*. 53–62.

SHNEIDERMAN, B. 2003. Promoting universal usability with multi-layer interface design. In *Proceedings of the Conference on Universal Usability (CUU'03).* ACM, New York, 1–8.

SHNEIDERMAN, B. 2007. Creativity support tools: Accelerating discovery and innovation. *Comm. ACM 50*, 12, 20–32.

SHU, N. C. 1989. Visual programming: Perspectives and approaches. *IBM Syst. J. 28*, 525–547.

SNYDER, C. 2003. *Paper Prototyping: The Fast and Easy Way to Define and Refine User Interfaces*. Morgan Kaufmann Publishers, San Francisco, CA.

STAHOVICH, T. F. 1998. The engineering sketch. *IEEE Intell. Syst. 13*, 3, 17–19.

STOLTERMAN, E. 2008. The nature of design practice and implications for interaction design research. *Int. J. Des. 2*, 1.

STOLTERMAN, E., MCATEE, J., ROYER, D., AND THANDAPANI, S. 2008. Designerly tools. In *Proceedings of the Design Research Society Biennial Conference*. article 116.

SUTHERLAND, I. E. 1963. Sketchpad, A man-machine graphical communication system. Ph.D. thesis, Electrical Engineering Department, Massachusetts Institute of Technology, Cambridge, MA.

THORNE, M., BURKE, D., AND VAN DE PANNE, M. 2004. Motion doodles: An interface for sketching character motion. In *ACM SIGGRAPH 2004 Papers*, J. Marks, Ed. ACM, New York, 424–431.

WALDO, J. 2000. *The Jini Specifications*, 2^{nd} Ed. Addison-Wesley Longman Publishing.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. SKETCH: An interface for sketching 3D scenes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, 163–170.

Received May 2009; revised July 2010; accepted October 2010