

Online Appendix to: Sketching Interactive Systems with Sketchify

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

Appendix A: Graphics, Animation and Freehand Sketching in Sketchify

A.1. INTRODUCTION

In this appendix we provide some details on how we implemented (and extended) freehand sketching within Sketchify. In the main text, in Section 6.1 we have already described the basic functionality of this environment. Here, we do not repeat this general description, but add more details about those elements of our sketching environment that we think could be useful for developers of similar systems. In the following sections, we describe individual components of Sketchify editor, including the following.

- Sketchify’s freehand drawing mode, used to create bitmap images,
- Active regions, used to create a range of interactive effects,
- Events and actions in Sketchify, used to support event-driven definition of interaction,
- Timers, used to produce advanced dynamic effects,
- Macros, used to group actions,
- Sketchify’s animation support,
- Formulas and templates in Sketchify, and
- Advanced options.

A.2. FREEHAND DRAWING MODE

In Sketchify, designers can create freehand sketches that consist of two types of elements: inactive elements, also called background images, and *active regions* (Figure 22).

Background images are created in the Sketchify drawing mode. From the designers’ point of view, the Sketchify drawing mode does not look much different from what might be expected from other simple image editors, as our environment supports most standard options for freehand drawing. A designer, for example, can use drawing tools, such as a brush or eraser, to create drawings. Working with multiple image layers is supported as well.

A.3. ACTIVE REGIONS

Most interactive effects in Sketchify are defined by means of active regions. An *active region* is a rectangular part in the sketch that can display drawings and text, but that can additionally capture user events and be graphically transformed (with transformations such as rotation, translation, or perspective mapping). An active region may embed other sketches that in turn contain background images and active regions.

A.3.1. Creating Images in Active Regions. The images (or drawings) that are associated with active regions can be created in several ways (Figure 23 and 24).

- By drawing them in the active regions image editor or in an external image editor,
- By importing them from a file or pasting them from the clipboard,

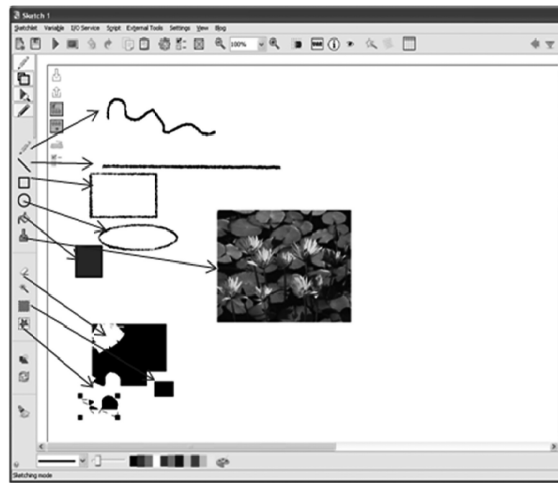


Fig. 22. The Sketchify image editor supports well-known bitmap image operations and tools, such as a brush and eraser to create drawings, importing an image from file, or opening the drawn image in an external editor for more complex image composition.

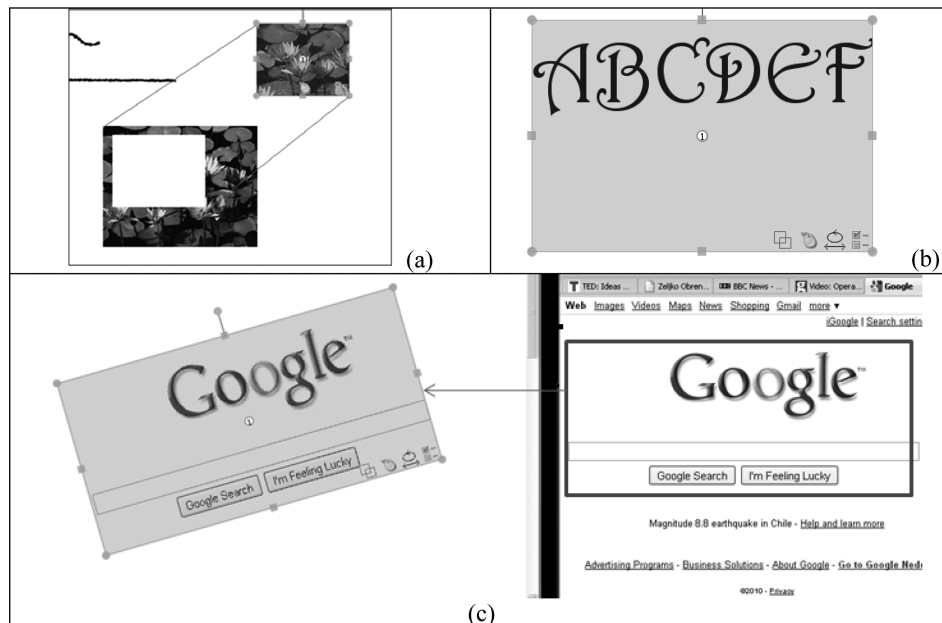


Fig. 23. Some of the ways to create an image in an active region: (a) extracting it from a background image; (b) as text or text file; (c) by capturing it from a part of the screen.

- By extracting a region from a background image created in drawing mode,
- By specifying a path or URL to an image file,
- By dynamically capturing them from screen,
- By defining a text to be rendered, or a more complex object including both graphics and text in a standardized format such as HyperText Markup Language (HTML) code, or Scalable Markup Language (SVG) code.

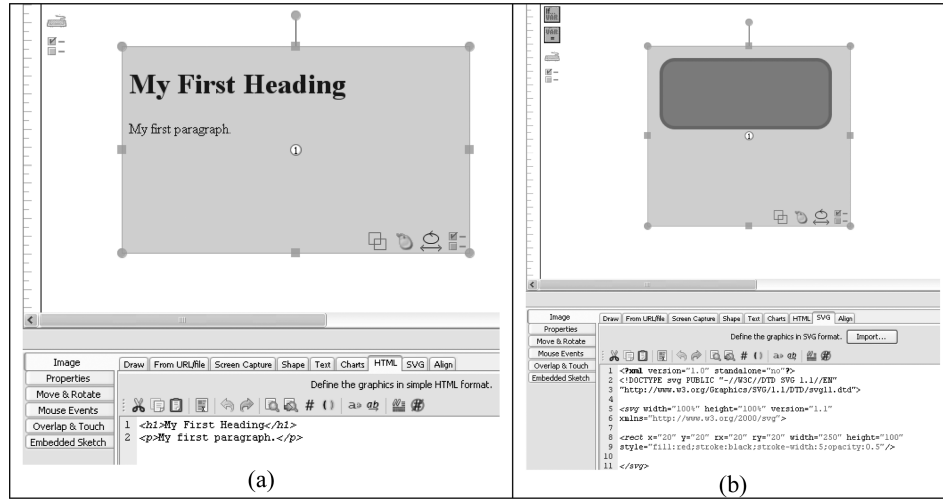


Fig. 24. Creating an image in an active region using markup languages: (a) from HyperText Markup Language (HTML) code; (b) from Scalable Vector Graphics (SVG) code.

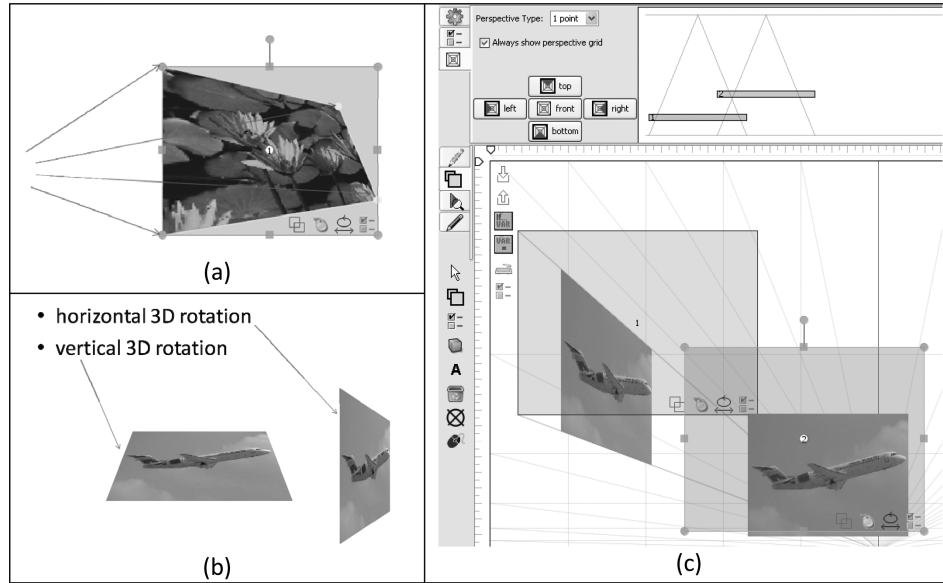


Fig. 25. Various forms of perspective graphical transformations of active regions: (a) manually specifying perspective points; (b) horizontal or vertical 3D rotation derives the perspective points from the specified angle; (c) a user interface for setting the perceptual depth of regions.

A.3.2. Graphical Transformations of Active Regions. Regardless of how an image is created, a designer can manipulate and transform such an image using a range of transformations (Table I). Using mouse or pen, a designer can change the position, size, orientation, and perspective points of active regions. Additionally, through a more advanced interface, a range of other transformations is also possible, such as changing transparency, or 3D rotation (Figure 25).

Table I. Some of the Possible Graphical Transformations of Active Regions

Transformation	Description
Position	
Position x	Horizontal position (in pixels)
Position y	Vertical position (in pixels)
Relative x	Relative horizontal position (between 0 and 1)
Relative y	Relative vertical position (between 0 and 1)
Trajectory position	Relative trajectory position (between 0 and 1)
Size	
Width	Width (in pixels)
Height	Height (in pixels)
Orientation	
Rotation	Rotation in the image plane (in degrees)
Image	
Transparency	Image transparency (between 0 and 1)
Visible Area	
Visible area x	X position of top, left corner of the visible area
Visible area y	Y position of top, left corner of the visible area
Visible area width	Width of the visible area
Visible area height	Height of the visible area
Motion	
Speed	Speed of the region (pixels per second)
Direction	Direction of motion if speed is defined (in degrees)
Pen	
Pen thickness	The thickness of the trace left when the region is moving
Shear	
Shear x	Horizontal shear transformation
Shear y	Vertical shear transformation
3D rotation	
Horizontal 3d rotation	Horizontal 3D rotation angle (in degrees)
Vertical 3d rotation	Vertical 3D rotation angle (in degrees)
Perspective	
Perspective depth	Perspective depth (between 0 and 1)

A.3.3. Embedding Sketches within Active Regions. Complete sketches, which may themselves contain active regions, can be embedded in another sketch through active regions (Figure 26). Embedded sketches may themselves be active, which means that they are capturing events, and are updating and reading variables. In order to avoid conflicts between different copies of the same sketch, each embedded sketch may be assigned a unique prefix and postfix for the variables that it depends on.

A.4. VARIABLES, FORMULAS AND TEMPLATES

Properties of active regions and sketches can also be specified indirectly through variables, formulas, and templates. Any property or expression that uses formulas and templates will be automatically evaluated when any of the variables in the formula/template is updated.

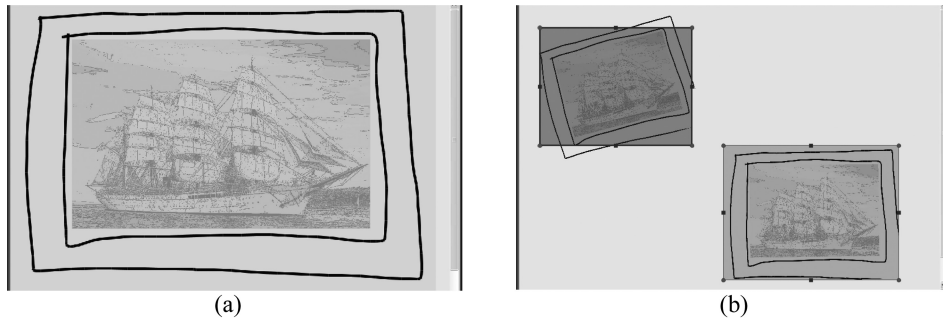


Fig. 26. Embedding sketches: the original sketch (a) is embedded twice in another sketch through two active regions (b).

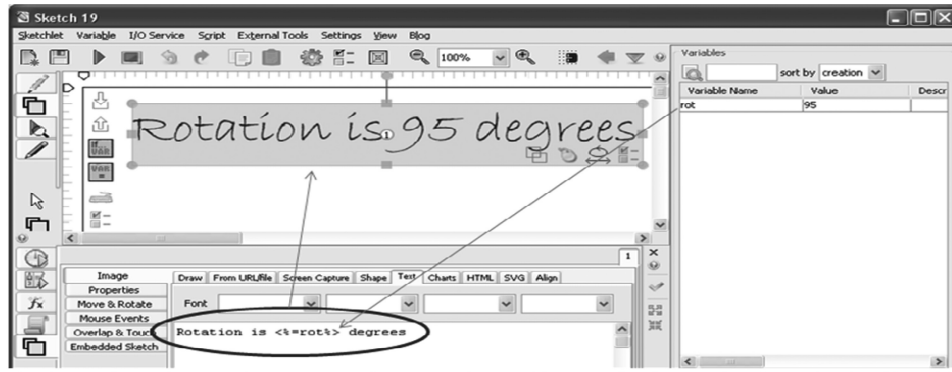


Fig. 27. Using string templates to set region properties. In this example, the template “Rotation is <%=rot%> degrees” is used to define the region text. The text will be refreshed and evaluated each time when the variable “rot” is updated.

A.4.1. Formulas. A designer can use formulas to derive values using diverse operators and functions, for example, what follows.

- $\text{=rot} * 3$, where the output value is calculated by multiplying the value of the variable “rot” by a factor of 3,
- $\text{=sqrt}(a^2 + b^2)$, where the value calculated is the square root of a sum of squares of values from variables “a” and “b”.

All common arithmetic operators are supported, as well as the conditional **if** operation.

A.4.2. Templates. Templates are a simple way to compose larger pieces of text by allowing variable names to be replaced by variable values, which are converted to text strings (Figure 27).

A.5. EVENTS AND ACTIONS

Sketchify enables an event-based description of interaction, where a designer can define the response to a range of interaction events (Table II). For each of these events one or more actions can be triggered (Table III).

Table II. Sketchify Events

Mouse Events	
<i>Continuous motion events</i> (when user drags the region)	<i>Discrete mouse events</i>
<i>Basic:</i> – position x, position y – rotation <i>Derived:</i> – speed of dragging – trajectory position (if defined)	<ul style="list-style-type: none"> • Left Button Click/Press/Release • Middle Button Click/Press/Release • Right Button Click/Press/Release • Double Click • Mouse Entry/Exit • Wheel Up/Down
Keyboard Events	
<ul style="list-style-type: none"> • Key pressed/released 	
Region Overlap Events (when two regions overlap)	
Variable Events (when variable is updated and has particular value)	
<ul style="list-style-type: none"> • Variable Updated • Variable Updated <condition> 	
Sketch Events	
<ul style="list-style-type: none"> • On sketch entry (when you open the sketch) • On sketch exit (when you close the sketch or move to another sketch) 	

Table III. Sketchify Actions

Sketch Actions	
Go to sketch	Transition to another sketch
Variable Actions	
Update variable	sets the variable to given value
Increment variable	increments current value of the variable by a given number
Append variable	appends string to the existing content of the variable
Timer actions	
Start timer	Starts a timer
Stop timer	Stops a timer
Macro Actions	
Start macro	Starts a macro (a list of actions)
Stop macro	Stops a macro

A.6. ANIMATION

Sketchify supports several ways to define animated effects, including:

- Flip-book-like animation,
- Properties animation, and
- Animation using timers (described in Section 7).

A.6.1. Flip Book Animation. Sketchify flip book animation is analogous to traditional flip book animation. A designer can create several image frames in an active region, and then animate them by defining the exposure time for each of the frames. A flip-book-like animation effect is created by looping through the images adopting the specified timing (Figure 28).

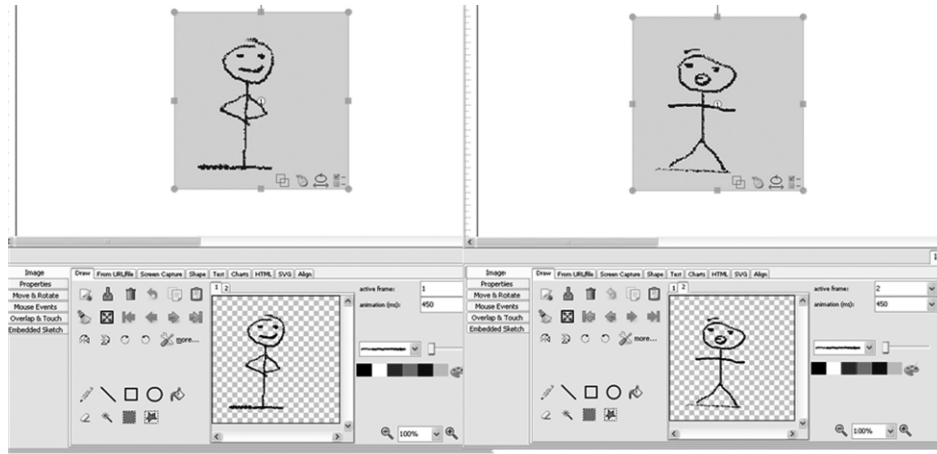


Fig. 28. Flip book animation in Sketchify. An active region can contain more than one image (drawn or imported). The (relative) exposure time for each of these images can be specified. A flip-book-like animation effect is created by looping through the image using the specified timing.

Table IV. Defining Properties for the Animation of an Active Region

Property	Animation Type	Start Value	End Value	Duration (sec)	Curve
Rotation	Loop Forever	0	360	1.0	linear
Transparency	Loop Forever	0.0	1.0	1.0	linear

In this example, two properties, rotation and transparency, are animated in a loop between their minimum and maximum values, with a cycle duration of 1 second.

A.6.2. Properties Animation. Sketchify supports simple animations for any of the numeric properties of sketches and active regions (see Table I for some of them). To define such animations, only few parameters need to be specified:

- Type of animation, which can be loop or pulse (once or forever),
- Start and end values of the property,
- Cycle duration, and
- Optional time curve.

For example, Table IV illustrates simultaneous animation of the rotation and transparency of an active region.

To simplify definition of this type of animations, we also provide an interface for interactively exploring various animation effects (Figure 29).

A.7. TIMERS

Timers facilitate the definition of more complex animation and interaction effects. A timer can cycle more than once, or work as a “pulsar” (a value increases from its minimum value to its maximum value in the forward cycle, while it decreases in the backward cycle). Timers introduce two elements:

- A timer variable interpolator, and
- A timeline with events.

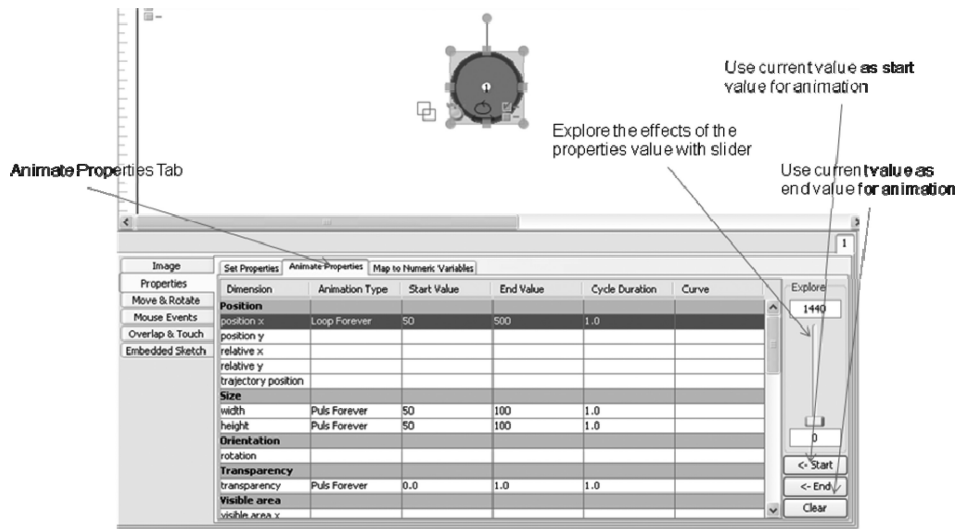


Fig. 29. A user interface for defining animation properties. In this example, each of the active region's transformation properties (such as position, rotation, transparency, perspective) can be animated by defining animation type, duration of the animation cycle, start and end values, as well as an optional time curve.

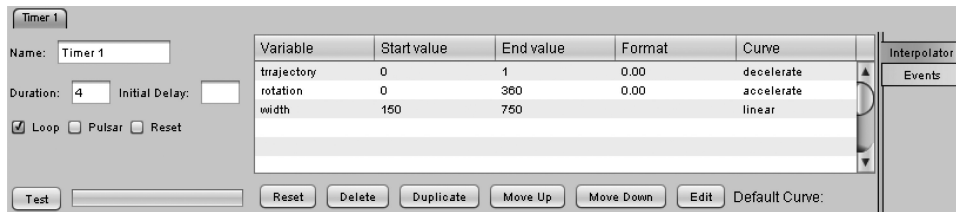


Fig. 30. A timer interpolator. During the timer cycle, the interpolator continuously changes one or more variables from their start to end values, and for each variable a different timer curve can be used.

A.7.1. Timer Variable Interpolator. Timer interpolators enable continuous updating of numeric variables (Figure 30). Timer interpolators are essential for animation effects. For example, we can define a timer with a cycle duration of 60 seconds and a resolution of one update per second to update the variable “orientation” from 0 to 360 degrees. This variable can, for instance, be used to control the orientation of an active region that simulates the handle of a clock.

Mapping between time and variable values can also be nonlinear, using timer curves (Figure 31). Timer curves enable nonlinear mappings between actual time and variable values in timers. When no timer curve is specified, the timer changes variable values linearly from start to end value. With a specified timer curve, however, the value transitions can have variable speed, for example, progressing fast at the beginning and slowing down at the end. A timer curve is defined independently from the timer itself. A timer can use a default timer curve which is used for all variable updates, but it can also use different curves for individual variables.

A.7.2. Timer Timeline and Events. Timer events enable a designer to define discrete events on the timer timeline (Figure 32), and associate one or more actions with such

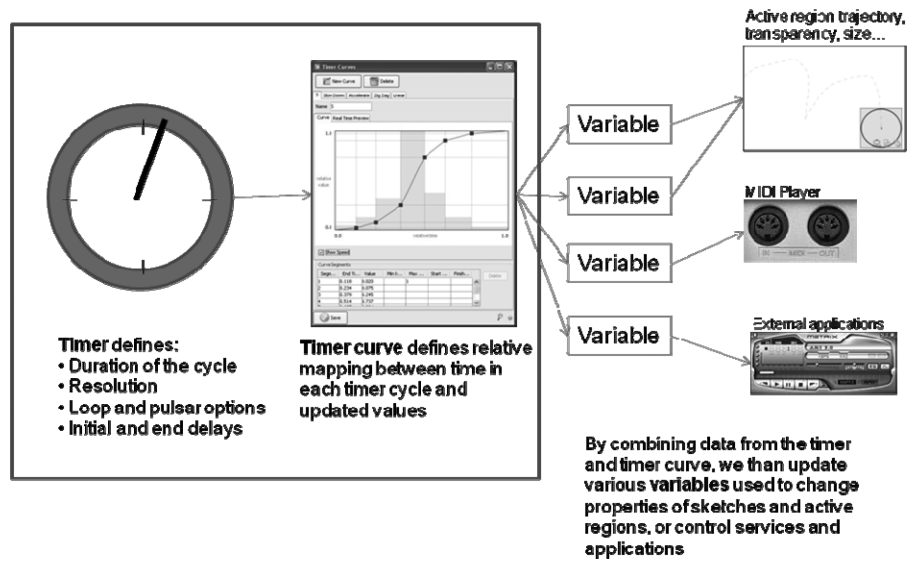


Fig. 31. Relation among timers, timer curves, and variables.

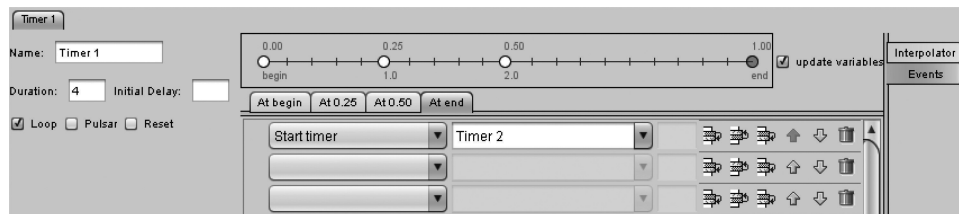


Fig. 32. Timer events: a designer can define a discrete point on the timeline, and define one or more actions that will be called when this point in time is reached during the timer execution.

an event. Points on the timeline are defined between 0 and 1, which means relative to the duration of the timer.

A.8. TRAJECTORIES

Sketchify exploits gesturing not only as a drawing modality, but also as a way to define a range of interactive effects. If a trajectory is defined for a region, the motion of that region is constrained to the sketched path. A designer can define primary and secondary trajectories as well as primary and secondary trajectory points for each region. If only one (primary) trajectory is defined, the region will be rotated to ensure that both region trajectory points stay on the trajectory (Figure 33(a)). If both primary and the secondary trajectories are defined (Figure 33(b)), the region will be rotated to ensure that the primary point stays on the primary trajectory, while the secondary point stays as close as possible to the secondary trajectory. This enables defining a range of simple mechanical simulations.

While defining the primary trajectory, the timing of the gesture is also recorded and made available as a template timer curve (Figure 34; also see the previous section).

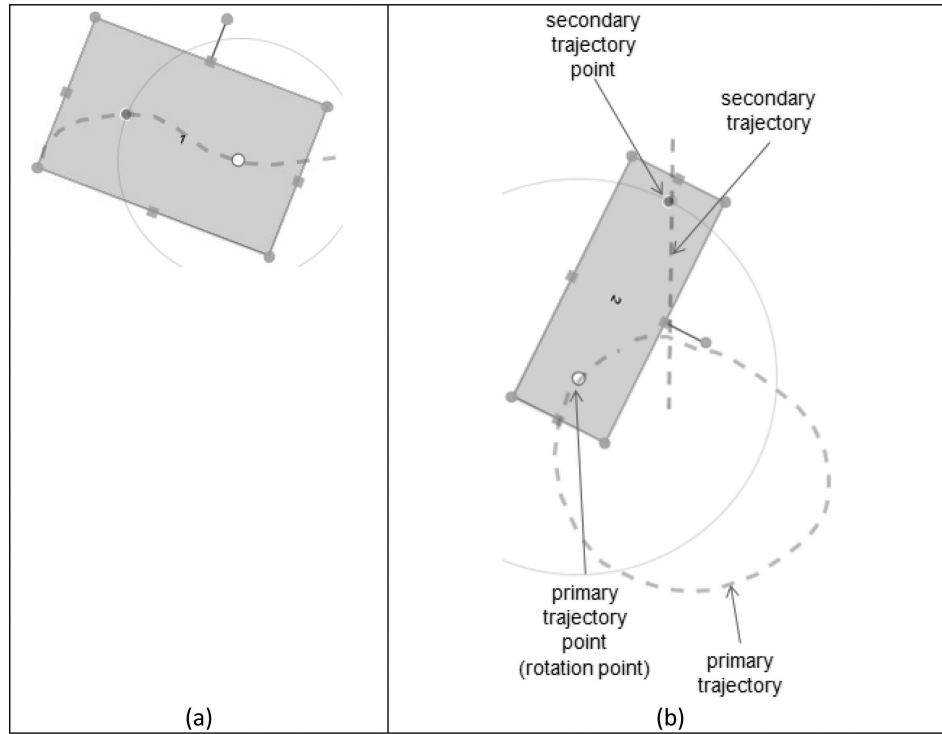


Fig. 33. Controlling the motion of the active region with trajectories: (a) when only a primary trajectory is defined, the region is rotated so that both identified points stay on the trajectory; (b) when a secondary trajectory is defined, the region is rotated so that the first point stays on the primary trajectory, and the second on the secondary trajectory.

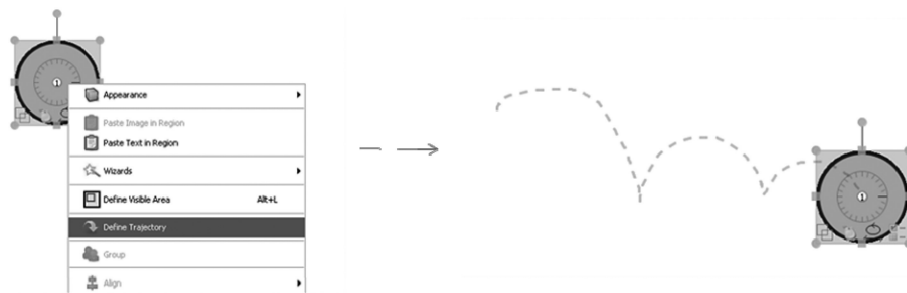


Fig. 34. Sketchify can record the trajectory of an active region. The resulting time curve can be used in the definition of timers and animations.

Table V. Control Structures in Sketchify Macros

IF <condition> END	Executes actions if a given condition is satisfied
PAUSE <time>	Pauses execution for a given time
WAIT UNTIL <condition>	Pauses execution until the condition (in terms of variable values, for example, “a > b”) is satisfied
WAIT FOR UPDATE <variable>	Pauses execution until a given variables is updated
REPEAT <n Forever> END	Repeat one or more actions forever or for a given number of times
STOP	Stops the execution of the macro

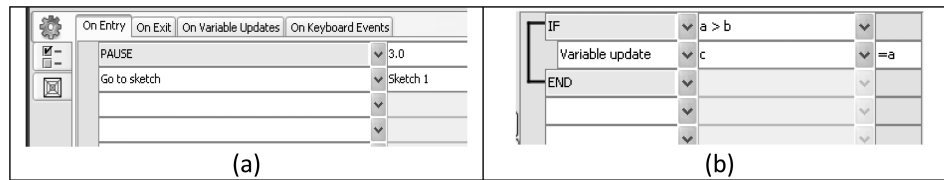


Fig. 35. Macros in Sketchify: (a) a built-in macro called on the sketch entry event; (b) a simple shared macro with conditional update of a variable.

A.9. MACROS

Sketchify macros enable creating more complex actions by grouping simpler ones (see Table III for list of available actions). In essence, a macro is a complex action with a list of successive commands that are triggered by a single event. Macros can include pauses in between actions, which facilitates control over dynamic effects (Figure 35(a)). Macros also support simple control structures (Table V), enabling the definition of conditional executions, loops, or synchronizing executions with variable updates (Figure 35(b)).

A.10. ADVANCED OPTIONS

Sketchify supports some advanced options, such as inherited sketches and mapping of a sketch to multiple display surfaces.

A.10.1. Sketch Inheritance. In order to support a more generic definition and reuse of existing sketches, we have introduced the concept of sketch inheritance. When one sketch inherits another, it automatically takes over its background image, its active regions, and all its settings. New elements and active regions can subsequently be added. As a special case of sketch inheritance, we also support the concept of a master sketch, which, if it exists, is inherited by all sketches (Figure 36).

A.10.2. Mapping Design Space to Display Space. When a sketch is executed, it will by default open in the same screen as the one it was created in, with a size that is identical to the one used in the design mode. However, we also offer the possibility for a more complex mapping between the design space and (one or more) display surfaces. More specifically, several display windows may exist at the same time, and for each of these display windows we can specify the part of the designed sketch that is shown in it. A transformation that can combine translation, scaling, rotation, and sheering can be applied (Figures 36 and 37). In this way a designer can use one sketch to control several displays by mapping different parts of the sketch to each display. This support may, for instance, be important when designing augmented reality applications, where

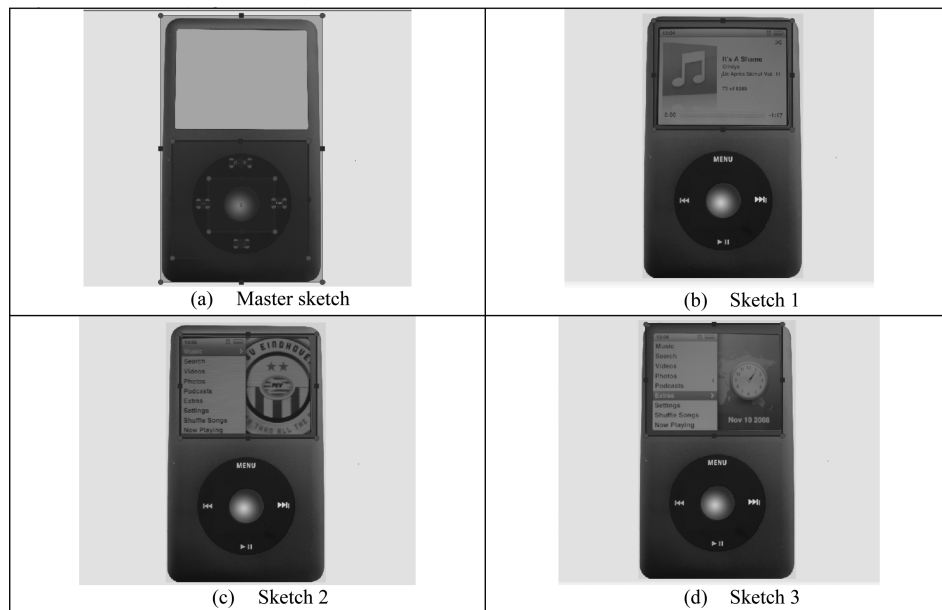


Fig. 36. A master sketch (a) defines a background image and active regions that are inherited in all other sketches (b, c, d) that add additional graphical elements and active regions.

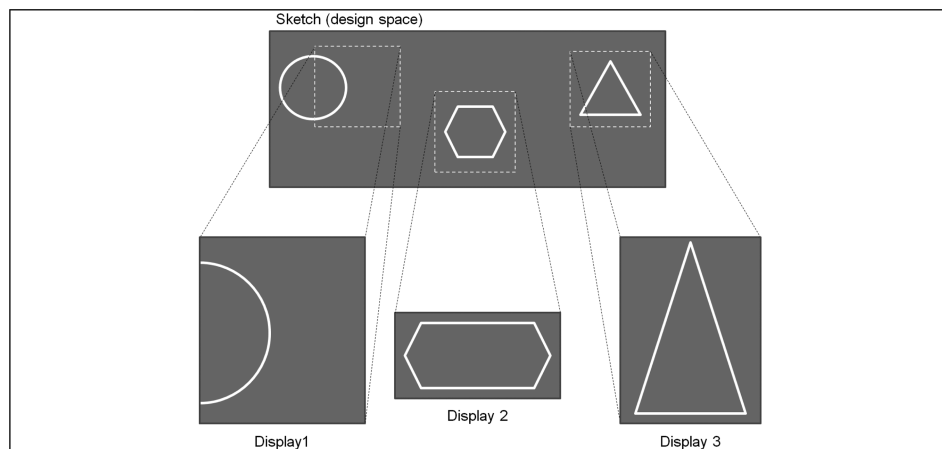


Fig. 37. Mapping design space to display space. A sketch can be presented on one or more screens, where for every screen we can define which part of the sketch is shown and how it is to be transformed.

several presentation spaces are often combined, such as a tabletop projection and a wall projection.

The mapping of design space to presentation space can also include image filters (Figure 39), as well as changes in the shape and transparency of the presentation window. Defining the shape and transparency of the window can be useful when a designer wants to combine Sketchify sketches with the visual output of existing applications without having to modify these latter applications.

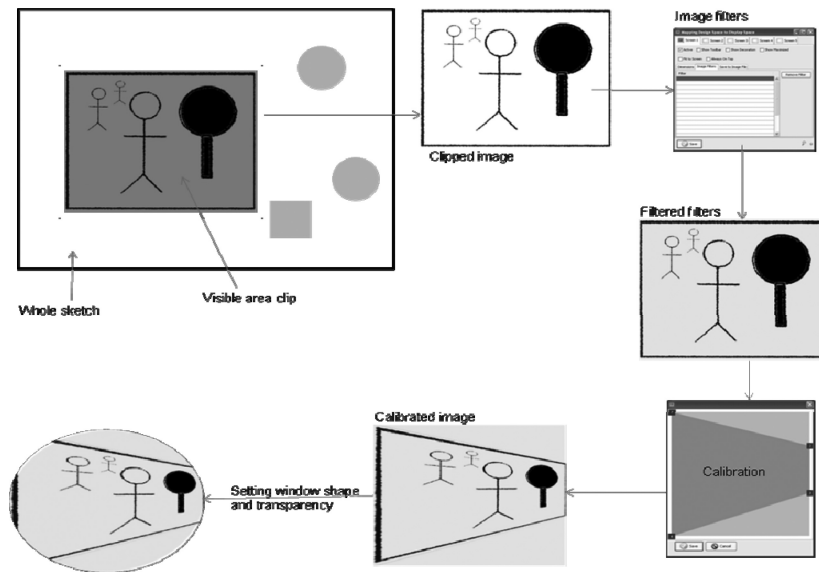


Fig. 38. Sketchify presentation pipeline. A selected region of the sketch is clipped, image filters are subsequently applied, and the filtered image is rendered using in perspective. The resulting image is presented in a window with a specified shape and transparency.

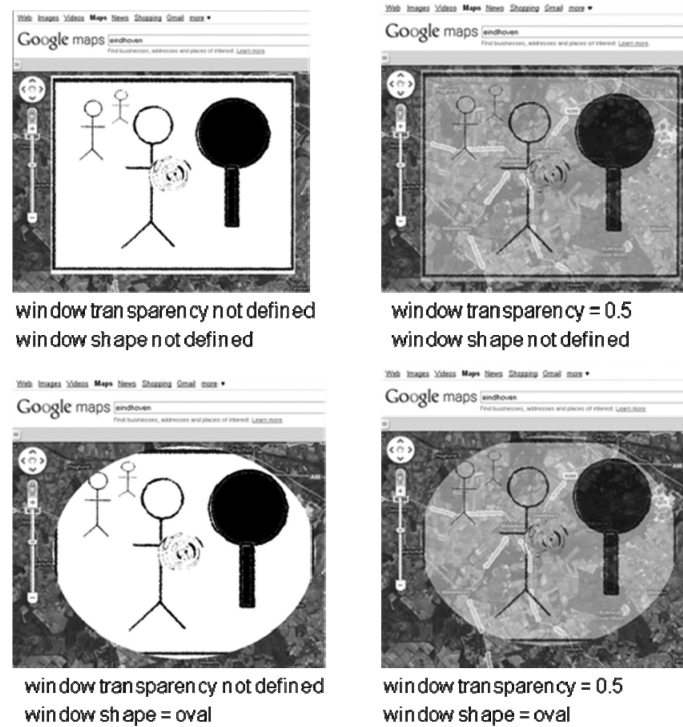


Fig. 39. A Sketchify sketch can be presented in windows with different shapes and transparencies, and overlaid on top of existing application windows.

Appendix B: Variables and Blackboard Implementation Details

B.1. INTRODUCTION

One of the most challenging implementation issues of Sketchify was coping with the complexity imposed by the requirement for integration of diverse components and environments. We had to solve many technical problems before we could combine the large variety of systems that were never meant to work together or even to be reused. In order to connect diverse elements, Sketchify uses middleware that is based on a loosely coupled blackboard coordination model, which in turn borrows ideas from context-enhanced collaborative applications and notification services [Edwards 2005; Obrenovic and Gasevic 2007; 2009]. Notification services are publish-and-subscribe infrastructures that allow applications to publish data items to zero or more subscribers. Often, these subscribers are applications that have registered to receive notifications. Such infrastructures are popular in collaborative applications and context-aware computing. For example, the Elvin system [Fitzpatrick et al. 1999] is a well-known pure notification service (without data storage), applied successfully to a number of collaborative applications. The Lotus Placeholder system [Dey 1999] is another notification service, which additionally incorporates a persistent data store that can be used by applications to manage shared data; changes to this data by any client result in notifications that are generated to other interested clients. A number of data repositories have models similar to these notification services. For example, tuplespace systems such as Linda [Gelernter 1985], the iROS [Johanson et al. 2002], and JavaSpaces and Jini [Waldo 2000] allow applications to store untyped tuples (collections of named data elements) in the model.

For Sketchify, we used the implementation of this model in the Adaptable Multi-Interface Communicator (AMICO) middleware,²¹ which has, for instance, been used successfully before to cope with the diversity of components in an application domain such as an interactive television [Obrenovic and Gasevic 2007]. One of the innovations in the Sketchify middleware which makes it suitable for the integration of many different components is the availability of a huge number of software integration interfaces. Sketchify middleware supports low-level socket-based networking interfaces, as well as many higher-level interfaces such as HTTP, XML-RPC, Open Sound Control (OSC), or SOAP. This facilitates reuse of existing services and components, as they do not have to be adapted to a common interface. Instead, the adaptation can be made using technology that is already supported by the component. The middleware is extendable and allows to easily add new service interfaces.

In this appendix, we describe the following elements of our middleware:

- the basic middleware functions,
- a number of available integration mechanisms,
- the user interface for working with variables,
- filtering and serialization of variables and the derivation of aggregate variables,
- the embedded Web server and the JSP engine.

B.2. MIDDLEWARE FUNCTIONS

Our middleware can be viewed as a blackboard system containing a list of variables [Obrenovic and Gasevic 2007]. Variables are untyped data objects, allowing the infrastructure to be very simple because the data model is highly generic. The infrastructure doesn't need to know the semantics or structure of the data it stores, which simplifies applying it to a wide range of applications. New applications can use existing data in

²¹<http://amico.sf.net/>

the model and add their own without conflicting with the infrastructure. All variables are placed in the same address space. The Sketchify middleware supports several functions common for message-oriented systems: UPDATE a variable, GET the value of a variable, REGISTER for notifications about a variable update, and DELETE a variable. To simplify the implementation of service adapters on top of this functionality, these functions are implemented in a fault-tolerant way: an update of a nonexisting variable creates a new variable, requests for a nonexisting variable returns an empty string rather than an error. We therefore do not introduce explicit actions for creating variables: variables are created by their first update in the a similar way as in the case of dynamically typed languages.

B.3. INTEGRATION MECHANISMS

One of the main innovations of Sketchify middleware is its ability to open up its basic functionality (update, get, register, and delete) through a huge number of service interfaces. These interfaces are needed to connect to other services and development environments. Most existing platforms provide integration mechanisms with many more constraints, and adapting software to work with such integration mechanisms can constitute a significant effort. We can group the interfaces that are being offered into three types.

- Loosely coupled service-oriented interfaces,
- Tightly-coupled plug-in interfaces,
- Auxiliary user-interface integration modules.

These interfaces are currently used to integrate software services, spreadsheets, and other environments.

Sketchify supports low-level TCP and UDP interfaces, where applications can update or read variables by exchanging string messages with our middleware, as well as many higher-level interfaces such as HTTP GET/POST, XML-RPC, OSC, or SOAP. For each of these interfaces, we developed adapters that map updates of variables to service calls and results of services to updates of variables (Figure 40). Our platform can be extended and allows for the addition of new service interfaces. This enables reuse of existing services and components, as they do not have to be adapted to a common interface, that is, adaptation can be made using the technology of the component, and developers can chose the interface they are most familiar with. We provide various data adapters for each of the service interfaces. For example, for handling SOAP-based XML data structures, we enable defining XML adapters. For XML-RPC and OSC and other service interfaces, we define mappings between variables and string equivalents of basic data types (such as integer or char) used by these interfaces.

When component code is not available, we enable limited integration of components through their user interfaces. We support three types of such interfaces.

- Command line interfaces, which we used to work with image processing tools, script editors, and to send a path to a spreadsheet file. Users can configure our environment and can use any of the programs that receive command line arguments and make calls based on variables updates.
- A keyboard and mouse simulator that can map variable updates to keyboard and mouse events.
- A screen capturer that can capture part of the screen at frequent time intervals, and that can save the result into a file, updating a variable with the corresponding file name.

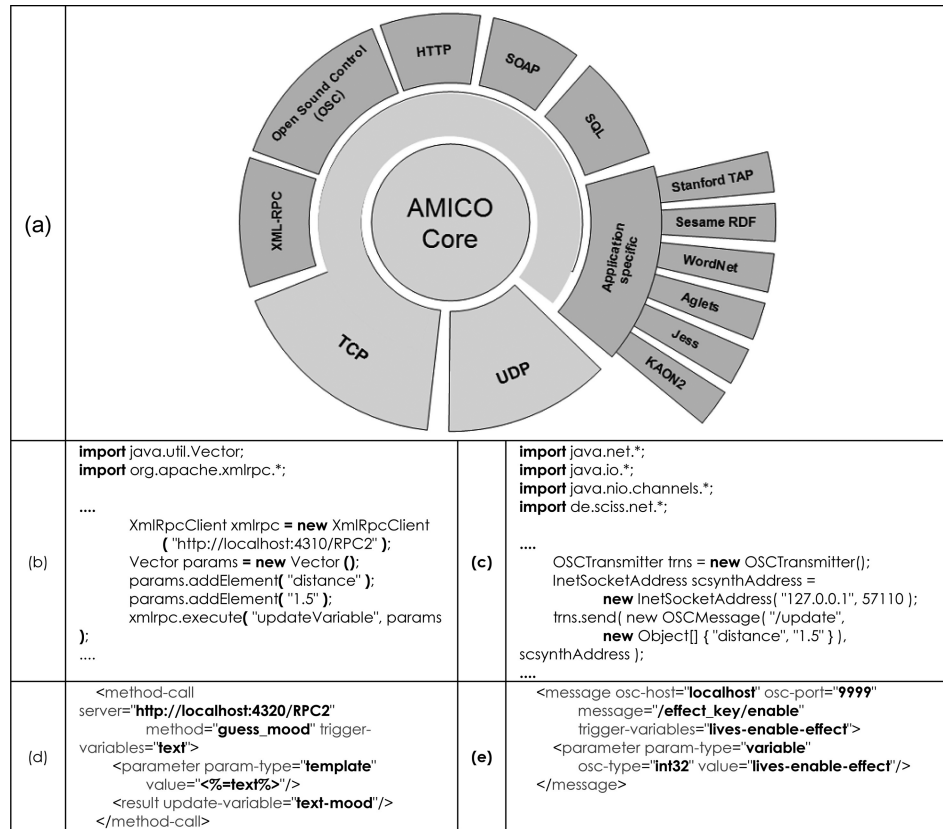


Fig. 40. Communication interfaces available within the integration infrastructure with some of the used components [Obrenovic and Gaswevic 2007]. The infrastructure currently supports a wide range of communication interfaces such as XML-RPC, OSC, URL, and SOAP, as well as many application-specific adapters (a). (b) and (c) show fragments of Java code for updating variables in the infrastructure using the XML-RPC and OSC interfaces, respectively. (d) and (e) show fragments of the adapter's configuration files for mapping variable updates into XML-RPC method calls and OSC messages, respectively.

B.4. USER INTERFACE FOR WORKING WITH VARIABLES

Sketchify variables can be accessed through a spreadsheet-like interface, making all data immediately visible and manipulable. For example, updating the value of the variable “tts-input” will cause the text-to-speech engine to pronounce the typed text. Results of various components will be visible as variable updates, such as a speech-command variable that represents the recognized speech. Through this interface the designer can directly observe and update variables, and this kind of interaction is very useful to explore and play with the functionality of software services. It can also serve as a Wizard-of-Oz backbone, where a human operator can update variables to simulate events in an environment.

In order to update and read variables within spreadsheets and scripts, designers have to write specific functions, which we used to define extensions. For example, to import a variable into a spreadsheet, we use the SKETCHIFY_READ formula, with the variable name as a parameter. This way of working is, however, slow and introduces a lot of errors due to misspelling of variable names. Therefore, in later versions of our tool, we included a pop-up menu that operates on the list of variables, and that allows to copy mathematical expressions with these variables to the clipboard, after

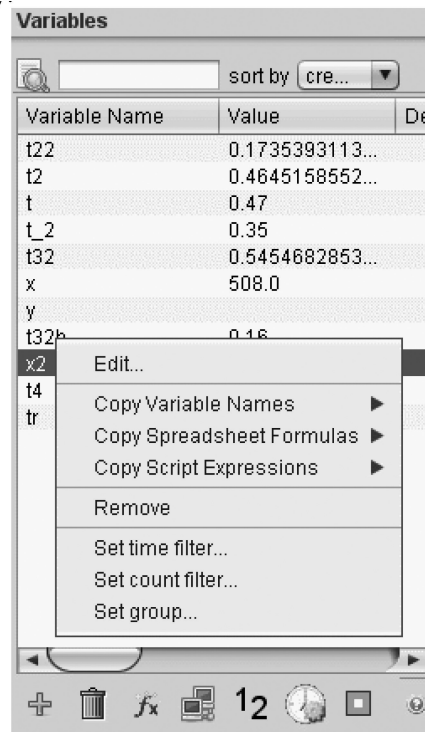


Fig. 41. User interface for working with variables. A user can modify the value of a variable, can add new variables, and can copy to the clipboard expressions that involve variable names, such as spreadsheet formulas or script expressions.

which they can be pasted into the spreadsheet or scripting environment (Figure 41). This modification has greatly reduced the number of errors and has improved the acceptance of the Sketchify tool. The variables also play a crucial role in creating a unified conceptual model of our loosely coupled framework, as a designer can work with very different elements and services in a similar way.

B.5. FILTERING, SERIALIZATION AND AGGREGATION OF VARIABLES

Filtering, serialization, and aggregation of variables are all aspects that serve to control the dynamics of the interaction (Figure 42). One of the problems we were facing, especially with software services, is that we were integrating components with very different timing characteristics. For example, some sensors can update variables more than fifty times per second, while spreadsheets are not able to process data this fast. To circumvent such problems, we introduced the concept of a “count filter.” For example, a count filter of 10 means that we will process every tenth update of a variable, ignoring the other nine. Sketchify furthermore enables serialization of variables, which means that sequential updates of a single variable are mapped to a sequence of new variables. Aggregate variables, in turn, are obtained by applying statistical functions on internally serialized variables. In this way, we can, for example, obtain a running average or standard deviation of a variable.

B.6. WEB SERVER AND JSP ENGINE

Sketchify also embeds a fully functional Web server and Java Server Pages (JSP) template engine, extended with support for reading and updating of Sketchify

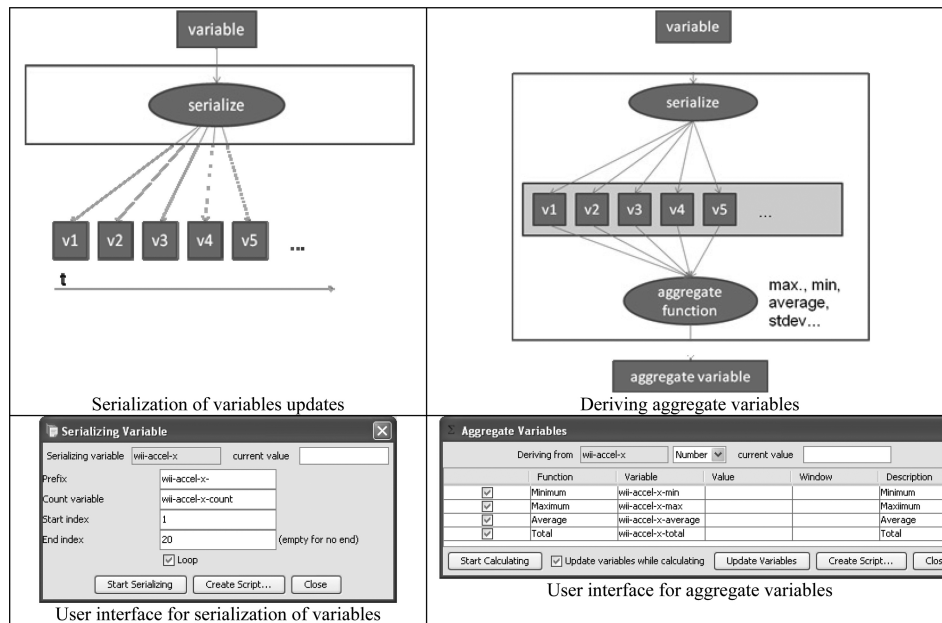


Fig. 42. Serialization and aggregation of variables. With serialization, a sequence of (child) variables are updated in response to the update of a single (parent) variable. For example, the first update of the parent variable will create and update variable “v1”, the second update will create and update variable “v2”, and so on. Aggregate variables are derived by applying mathematical function on a sequence of variables.

variables. In this way, a designer experienced in Web markup languages, such as HTML, Scalable Vector Graphics (SVG), or VRML, can write simple declarative templates for generating content that can be visualized in a Web browser. Figure 43 gives a simple example of a VRML page that represents a simple box, where the X component of the box size is taken from the variable “a”. This support is convenient for more advanced prototyping. All that is needed to make this functionality available is to create or import a file describing a Web page in the same directory as the sketches. The designer will be offered the option to open this file in a Web browser (see Figure 43(c)).

Appendix C: Sketchify End-User Development Tools

C.1. INTRODUCTION

Sketchify provides access to several end-user programming tools that can help designers to define more complex interaction behaviors in their sketches. We currently support OpenOffice.org CALC spreadsheets, as well as several higher-level scripting languages including Javascript, Python, BeanShell, Groovy, Ruby, TCL, Sleep, Haskell, and Prolog.

C.2. SPREADSHEETS

Spreadsheets were included in Sketchify by means of add-ins for OpenOffice.org CALC and Microsoft Excel [Obrenovic and Gasevic 2008]. From the end-user point of view the add-ons constitute only a few additional functions that are accessible as spreadsheet formulas (Table VI).

The introduced functions allow users to update or read all variables of any service connected to Sketchify. For example, the expression `SKETCHIFY.WRITE(“spelling”; B10)` is evaluated every time when cell B10 is updated, and calls the Google spelling

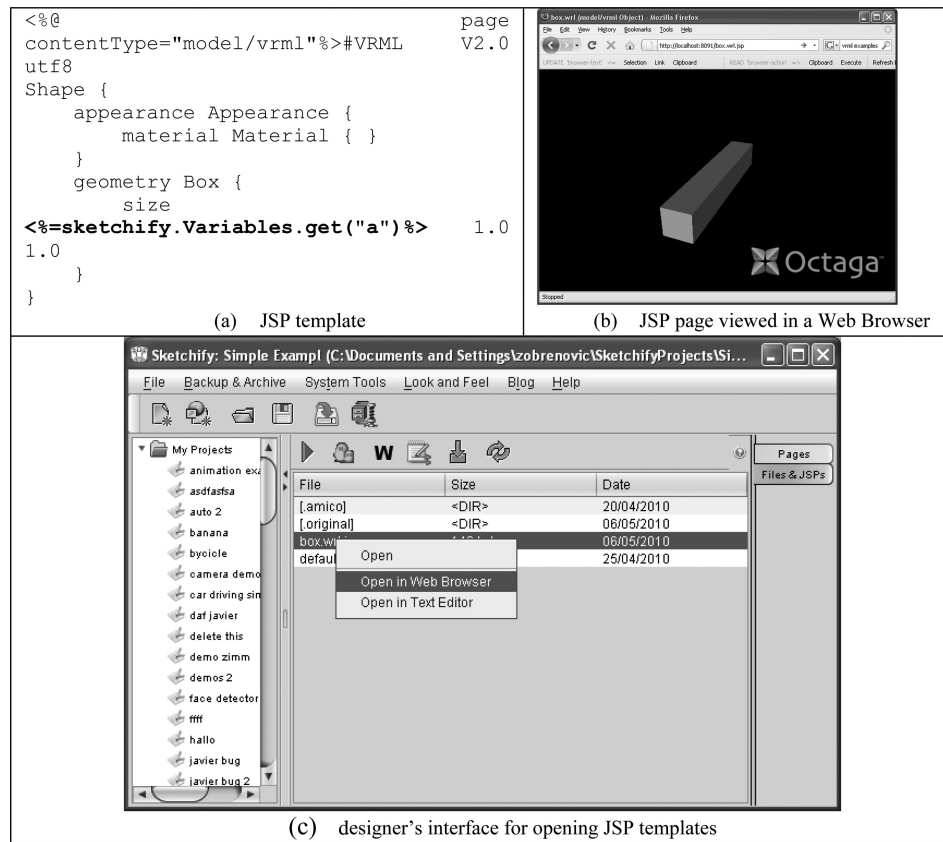


Fig. 43. Example JSP page.

checker service (which is triggered by updating the Sketchify variable “spelling”); while expression `SKETCHIFY_READ(“spelling-suggestion”)` obtains the current value of the variable “spelling-suggestion”, which is updated by the Google spelling checker adapter. Reading a variable once also registers an application for notifications about future changes of this variable.

Spreadsheets run as separate process and connect to our middleware through TCP and UDP interfaces.

C.3. SUPPORT FOR SCRIPTING LANGUAGES

Scripting languages are relatively easy to learn by developers as they use typeless approaches to achieve a higher level of programming. This tends to result in much more rapid application development than when using low-level programming languages. However, there is a huge diversity of scripting languages, each being supported by a significant community. Therefore, instead of choosing one specific language, we decided to support most popular scripting languages, including:

- (1) Javascript, a dynamic, weakly typed prototype-based language,
- (2) Python, a very high-level programming language supporting multiple programming paradigms (object oriented, imperative, and functional),
- (3) BeanShell and Groovy, two Java-based scripting languages,
- (4) Ruby, a dynamic, reflective, general-purpose object-oriented programming language,

Table VI. Spreadsheet Extensions and Their Corresponding Functions

Function
SKETCHIFY_WRITE (<i><variable-names></i> , <i><values></i>) Updates Sketchify variables with the specified values (returns the same value, or the last value in case a range of cells is specified).
SKETCHIFY_READ (<i><variable-name></i>) Registers for notifications of a Sketchify variable with a given name and updates the spreadsheet every time when a new value is received. Our spreadsheet extension creates a thread that listens for notifications of registered variables, and propagates the received values to the spreadsheet formulas that use this value.
SKETCHIFY_WRITE_DELAYED (<i><variable-names></i> , <i><values></i> , <i><delays></i>) Updates Sketchify variable(s) with given values, after (a) given delay(s). For example, SKETCHIFY_WRITE_DELAYED ("A", "test", 2.5) will update the variable A with the value "test" after 2.5 seconds. SKETCHIFY_WRITE_DELAYED (A1:A10, B1:B10, C1:C10) will update variables with names defined in cells A1 to A10, with values defined in cells B1 to B10, with delays defined in cells C1 to C10 (i.e., the function will first wait for a period defined in C1 and then update a variable with the name defined in cell A1 with the value from cell B1, then it will wait for a period defined in cell C2 before updating variables with the name defined in cell B2, and so on). The function returns the values as they are updated (in our example, it returns B1, B2 ... B10).
SKETCHIFY_READ_LOOP (<i><variable></i> , <i><cell-or-row></i> , <i><cel/row-id></i> , <i><start-value></i> , <i><end-value></i> , <i><step></i>) Maps sequential updates of a Sketchify variable into a spatial update of spreadsheet cells. Updates are performed incrementally within a given row or column, with a given step. For example, SKETCHIFY_READ_LOOP ("var1", "column", "A", 5, 10, 1), will map updates of variable var1 to a range of cells; the first update will update cell A5, the second update to A6, etc.

JavaScript (a)	Prolog (b)
<pre>function variableUpdated(name, value, oldValue) { if (name == "spelling-suggestion") { if (value == "") { sketchify.update("tts-text", "No suggestions"); } else { sketchify.update("tts-text", value); } } }</pre>	<pre>variable("spelling-suggestion", "") :- sketchify_update("tts-text","No suggestions"). variable("spelling-suggestion", V) :- sketchify_update("tts-text", V).</pre>

Fig. 44. Examples of scripts written in Javascript and Prolog. Both scripts produce the same effect: sending the result of a spelling suggestion Web service to a local text-to-speech service when the variable "spelling-suggestion" is updated.

- (5) TCL, a popular tool command language,
- (6) Sleep, a procedural scripting language inspired by Perl and Objective-C,
- (7) Haskell, a standardized purely functional programming language with nonstrict semantics, and
- (8) Prolog, a logic programming language.

Except for the support for Prolog, which is based on the JLog project,²² all scripting support is based on the Java Scripting Project.²³ Our middleware runs scripting engines for each of these languages, extending each language with functions for updating and reading variables (i.e., functions UPDATE and GET). To receive notifications about variable updates (i.e., the REGISTER function), a script needs to contain a hook function called variableUpdated (see Figure 44(a)), which can be called by the middleware when a given variable is updated. For Prolog, which is a nonprocedural scripting language, we introduced special predicates for updating, reading, and receiving notifications about variable updates (see Figure 44(b)).

²²<http://jlogic.sourceforge.net/>

²³<https://scripting.dev.java.net/>

We also support XSLT scripts implemented using standard Java XML libraries. These scripting extensions allow developers to use (a mixture) of different programming modes, such as declarative programming, object-oriented programming, or logic programming, when creating their services.

Appendix D: Simplified Software Services within Sketchify



D.1. INTRODUCTION

In this appendix we list the major simplified software services that have up to now been included within Sketchify. Our software services are real but “trimmed down” input/output devices and services that have been extracted from diverse domains. Such services can bring components from various domains within reach of designers, and allow them to incorporate real functionality within a design already at an early stage, without requiring programming skills on the side of a designer. The obvious advantage of this is, of course, that the designer and potential end-users can develop a better “feel” for the application being designed.

Currently supported software services include camera-based face and motion detectors, text-to-speech engines and speech recognizers, VRPN devices (such as 3D trackers and buttons), MP3 and MIDI players, Web services (such as a Google spelling checker and search engine), a Wordnet definition service, and many others. Figure 45 lists these services, and provides references to sections in this appendix where each of these software services is described. For each software service, we describe the designer interface towards these services, which consists of one or more variables connected to the service. Source-code examples for many of these services are available as part of our open-source package.²⁴

D.2. SPEECH SERVICES

In order to enable sketching of speech-enabled interactive systems, Sketchify has been connected to several open-source Text-To-Speech (TTS) engines and speech recognizers, including an English speech recognizer based on Sphinx-4²⁵ the FreeTTS English TTS engine,²⁶ the NEXTENS Dutch TTS engine,²⁷ and the Mary TTS engine that currently supports English, German and Tibetan.²⁸ The preceding table illustrates the variables being used to connect to FreeTTS and to the Sphinx-4 systems.

	Direction *	Variables	Description
FreeTTS text-to-speech engine 	←	tts-input	Text to be pronounced.
	→	tts-status	Status of the engine: ‘loading’, ‘ready’, ‘talking’
Sphinx-4 speech recognizer 	→	speech-command	Text produced by the recognizer
	→	sphinx4-status	Status of the engine: ‘loading’, ‘ready’, ‘talking’

← = read by the service (parameters of a service); → = updated by the service (results of a service)

²⁴<http://sketchify.sf.net/>

²⁵<http://cmusphinx.sourceforge.net/sphinx4/>

²⁶<http://freetts.sourceforge.net/>

²⁷<http://nextens.uvt.nl/>

²⁸<http://mary.dfki.de/>

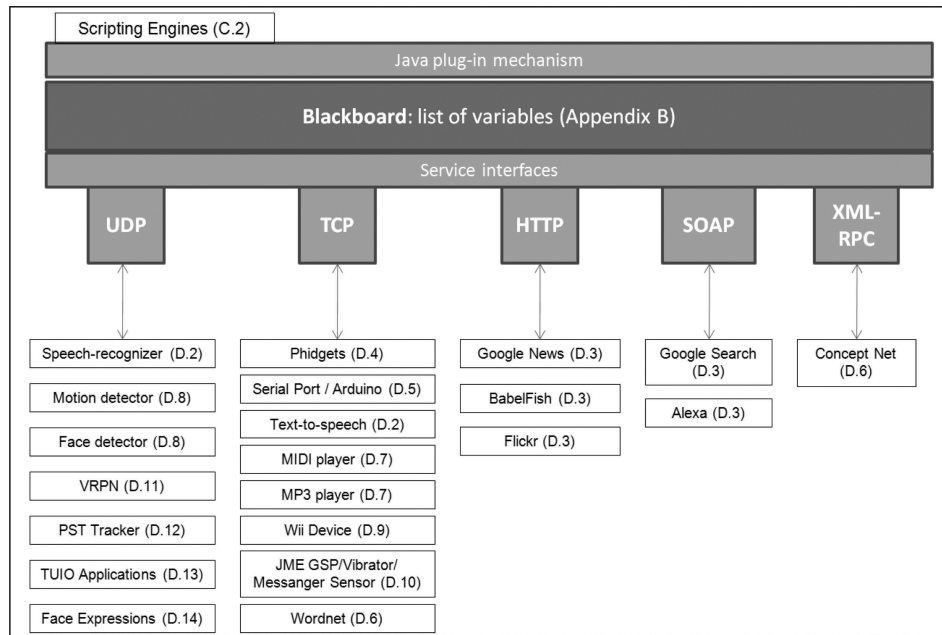



Fig. 45. Sketchify's simplified software services, interfaces used to connect to the variables, and section numbers were each service is described.

D.3. WEB SERVICES

Our platform also allows to connect to various Web services, including the Google SOAP search service and spelling checker,²⁹ Google News search HTTP RSS Web service,³⁰ Yahoo Babel fish HTTP HTML translation service³¹ the Flickr image uploader³² and the Alexa statistical Web service.³³ Our platform allows for easy adaptation to other Web services.

	Direction	Variables	Description
Google News Search Web Service 	←	google-news-query	User query
	→	google-news-title-<n>	Title of the news item <n> (n = 1..5)
	→	google-news-date-<n>	Date of the news item <n> (n = 1..5)
	→	google-news-link-<n>	Link to the news item <n> (n = 1..5)
	→	google-news-status	Status of search ('ready', 'working', 'finished')

(continued)

²⁹<http://code.google.com/apis/soapsearch/>




³⁰<http://news.google.com/>

³¹<http://babelfish.yahoo.com/>

³²<http://www.flickr.com/services/api/>

³³<http://aws.amazon.com/>




(Continued)

	Direction	Variables	Description
BabelFish translation Web service 	←	babelfish-input-text	Text to be translated
	←	babelfish-input-language	Language of user text ('en', 'nl', ...)
	←	babelfish-output-language	Language of translation ('en', 'nl', ...)
	→	babelfish-translation	Translated text
	→	babelfish-status	Status of translation service
Flickr Image Upload Web Service 	←	flickr-upload-file	Path to local image file
	←	flickr-upload-photo-title	Image title
	←	flickr-upload-photo-description	Image description
	←	flickr-upload-photo-family	Related to user's family ('yes'/'no')
	←	flickr-upload-photo-friend	Related to user's friends ('yes'/'no')
	←	flickr-upload-photo-public	Public image ('yes'/'no')
	←	flickr-upload-tags	Tags (comma separated)
	←	flickr-upload-tag-<n>	Giving separate tags
	→	flickr-uploader-status	Status of service
	→	flickr-uploaded-photo-id	ID of uploaded photo
Alexa Amazon Web Service 	←	alexa-info-url	URL for which you ask statistics
	→	alexa-status	Status of service
	→	alexa-info-url-rank	URL rank
	→	alexa-info-url-page-views-3-months	Number of views of the URL for last 3 months
	→	alexa-info-url-page-views-1-month	Number of views of the URL for last month
	→	alexa-info-url-page-views-1-week	Number of views of the URL for last week
	→	alexa-info-url-page-views-1-day	Number of views of the URL for last day

D.4. PHIDGETS




Phidgets³⁴ is a system consisting of low-cost electronic components and sensors that are controlled by a personal computer, connected to the PC using the Universal Serial Bus (USB). Phidgets consists of a set of plug-and-play building blocks for low-cost USB sensing and control from your PC. Our adaptation is based on the Phidgets Java API.

³⁴<http://www.phidgets.com/>

	Direction	Variables	Description
Phidget Sensors and Actuators 		phidget-<deviceid>-value	
		phidget-<deviceid>-input-<n>	

D.5. SERIAL PORTS AND ARDUINO

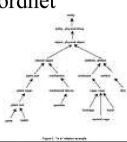













Sketchify can receive and send data to serial ports, enabling designers to work with devices and platforms such as Arduino.³⁵

	Direction	Variables	Description
Serial Interface 		serial-send	data to be send to a serial port
		serial-receive	data received from a serial port

Other Sketchify variables can be directly updated from a serial port if the data are written in a format “var=value”.

D.6. SEMANTIC SERVICES

We also support two links to semantic services: a Wordnet definition service³⁶ and ConceptNet,³⁷ a common-sense reasoning framework.

	Direction	Variables	Description
Wordnet 		wordnet-query	Word or phrase
		wordnet-definition	Definition of word/phrase
		wordnet-hyponym-<n>	Hyponyms of the word/phrase
		wordnet-hypernym-<n>	Hypernyms of the word/phrase
		wordnet-status	Status of service
ConceptNet 		text-context	Text to be analyzed
		text-context-mood	Estimated mood of text
		text-context-concept	Concepts found in text
		text-context-topic	Topics found in text
		text-context-lemmatise	Lemmatized text
		text-context-summarize	Text summary
		text-context-chunk	Text chunks

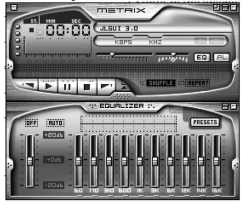
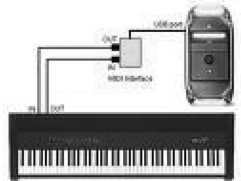
³⁵<http://www.arduino.cc/>

³⁶<http://wordnet.princeton.edu/>

³⁷<http://web.media.mit.edu/~hugo/conceptnet/>



D.7. MUSIC SOFTWARE SERVICES

We currently support two music output tools: an MP3 music player based on the `jlGUI` open-source Java MP3 player,³⁸ and a MIDI player implemented using standard Java audio libraries.

	Direction	Variables	Description
	←	mp3-song	URL or path of audio file to be played
	←	mp3-command	Playback commands: 'start', 'stop', 'pause', 'next', 'previous', 'eject'
	←	mp3-volume	Sound intensity
	←	mp3-equalizer	Main equalizer level
	←	mp3-equalizer-<channel>	Equalizer level per channel
	←	midi-note	A note to be played in format "<duration> <velocity> <tone>"
	←	midi-instrument	Music instrument being played

D.8. COMPUTER VISION SOFTWARE SERVICES

We have adapted several computing vision modules, based on the OpenCV Computer Vision Library³⁹ including a motion detector and a face detector.

	Direction	Variables	Description
	⇒	motion-intensity	Intensity of motion derived from the difference between successive images.
	⇒	number-of-faces	Number of faces detected: 0, 1, 2, ...
	⇒	face-<id>-x1	Left
	⇒	face-<id>-y1	Top
	⇒	face-<id>-x2	Right
	⇒	face-<id>-y2	Bottom

D.9. WII REMOTE

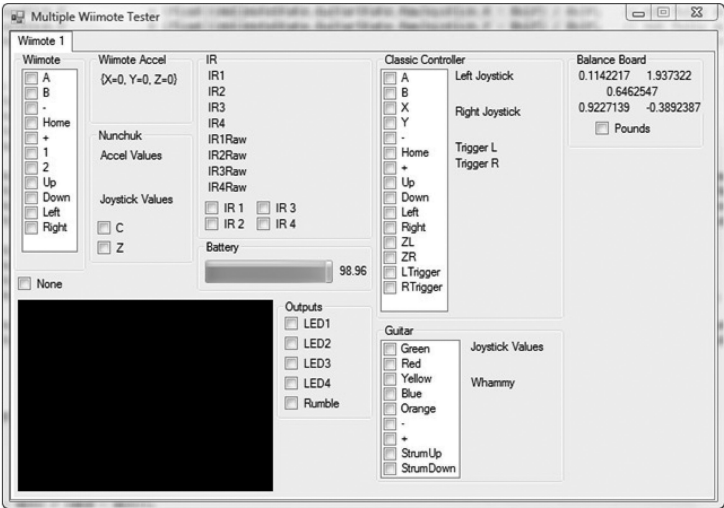
Wii Remote,⁴⁰ which connects to a PC using a Bluetooth link, is a complex sensing platform. It can track infra-red sources, and contains three acceleration sensors, various buttons, a vibrator, a simple speaker, and some status LED diodes. It can also be used















³⁸<http://www.javazoom.net/jlgui/jlgui.html>

³⁹<http://opencvlibrary.sourceforge.net/>

⁴⁰http://en.wikipedia.org/wiki/Wii_Remote

to connect more devices, such as Wii Nunchuk, which contains a joystick and more buttons. Other related devices, such as Wii Fit, can also be used. Our Wii software service is based on the C# demo programs that come with WiimoteLib.⁴¹














	Direction	Variables	Description
 Wii Remote  Wii Nunchuk  Wii Fit		wii-<wii-id>-accel-x	X-axis acceleration (in Gs)
		wii-<wii-id>-accel-y	Y-axis acceleration (in Gs)
		wii-<wii-id>-accel-z	Z-axis acceleration (in Gs)
		wii-<wii-id>-ir-<object-id>-x	X position of a tracked infrared object (0..1.0)
		wii-<wii-id>-ir-<object-id>-y	Y position of a tracked infrared object (0..1.0)
		wii-<wii-id> -<button-id>-state	State of each Wii Remote button ('True' or 'False')
		wii-<wii-id>-led-<led-id>-status	Status of led diodes on Wii ('on' or 'off')
		wii-<wii-id>-vibrate-ms	Causes the Wii Remote to vibrate for a given time (in milliseconds)
		wii-<wii-id>-fit-<led-id>-status	The status of LED diodes on Wii Fit
		wii-<wii-id>-guitar-<led-id>-status	The status of LED diodes on Wii Guitar Device
		wii-fit-balance-<n>	Data from one of four balance board sensors

⁴¹<http://www.codeplex.com/WiimoteLib>








D.10. JAVAME MOBILE PHONE SOFTWARE SERVICES

Many mobile phones support the Java Platform Micro Edition (Java ME,⁴²) which provides programmers with access to phone resources. We have developed several mobile phone software services using standard JavaME libraries, including a GPS sensor service, a photo capturer, an SMS messenger, and a controller for the mobile phone vibrator.

JavaME Mobile Phone	Direction	Variables	Description
		gps-status	Status of GPS module
		gps-latitude	Detected latitude
		gps-longitude	Detected longitude
		gps-last-latitude	Last detected latitude
		gps-last-longitude	Last detected longitude
		gps-iteration	Number of updates of GPS position
		mobile-vibrate-command	Command (millis) to cause vibration
		mobile-vibrate-status	Status of the vibrator
		sms-number	SMS number
		sms-message	Text of SMS message
		sms-status	Status of SMS module

D.11. VRPN MODULES

The Virtual Reality Peripheral Network (VRPN,⁴³) is a programming library that implements a standardized and network-transparent interface between application programs and physical devices used in Virtual-Reality (VR) systems (such as 3D trackers). Any available VRPN device can be connected to our platform. For example, we have been using the Intersense IS-600 tracker,⁴⁴ an infrasonic sensor that tracks the position of objects in 3D. It can track several objects at the same time, and for each object it can detect its position (x, y, z), and when used in combination with the inertial cube, also its orientation.

IS600 tracker	Direction	Variables	Description
		Is600-<id>-position-1	X coordinate
		Is600-<id>-position-2	Y coordinate
		Is600-<id>-position-3	Z coordinate
		Is600-<id>-orientation-1	Axis of rotation
		Is600-<id>-orientation-2	
		Is600-<id>-orientation-3	
		Is600-<id>-orientation-4	Angle









⁴²<http://java.sun.com/javame/technology/>

⁴³<http://www.cs.unc.edu/Research/vrpn/>

⁴⁴<http://www.isense.com/>







D.12. PST OPTICAL TRACKER

The optical tracker of Personal Space Technologies (PST,⁴⁵) can track 3D objects that are tagged with infrared markers.

	Direction	Variables	Description
Personal Space Technologies (PST) Optical Tracker 		pst-<id>-position-1	X coordinate
		pst-<id>-position-2	Y coordinate
		pst-<id>-position-3	Z coordinate
		pst-<id>-orientation-1	Axis of rotation
		pst-<id>-orientation-2	
		pst-<id>-orientation-3	
		pst-<id>-orientation-4	Angle





D.13. TUIO DEVICES

Sketchify can also work with applications that support TUIO protocol.⁴⁶ Such applications can track several objects or cursors and send data about their state. This means that Sketchify can receive input from tangible multitouch surfaces (such as iPhone or iPad), or from vision-based multiobject tracking systems (such as reacTIVision,⁴⁷).

	Direction	Variables	Description
TUIO Application 		tuio-object-<id>-x	Horizontal position of a TUIO object
		tuio-object-<id>-y	Vertical position of a TUIO object
		tuio-object-<id>-a	Orientation (angle) of a TUIO object
		tuio-cursor-<id>-x	Horizontal position of a TUIO cursor
		tuio-cursor-<id>-y	Vertical position of a TUIO cursor

D.14. FACE EXPRESSIONS

We support a simple face expression animation module based on the Expression toolkit:⁴⁸ an open-source procedural facial animation system. In our adaptation, the face animation runs in a separate window, and through variables a designer can set basic and complex facial expressions as well as define the “mood” of the character.

	Direction	Variables	Description
Face Expressions 		face-expression	ID of the face expression to be animated (1..41)
		face-composite-expression	ID of one of 12 complex face expressions to be animated
		face-mood	Face mood during animation ('happy', 'sad', 'angry', 'scared', 'tired', 'skeptical')

⁴⁵<http://www.ps-tech.com/>

⁴⁶<http://www.tuio.org>

⁴⁷<http://reactivision.sourceforge.net/>

⁴⁸<http://expression.sourceforge.net/>