Bettina Conradi, Balázs Serényi, Miriam Kranz, Heinrich Hussmann (2010):
SourceBinder: Community-based Visual and Physical Prototyping.
In Steffen Budweg, Sebastian Draxler, Steffen Lohmann, Asarnusch Rashid, Gunnar Stevens (Eds.), International Reports on Socio-Informatics (IRS
Proceedings of the 2nd International Workshop on Open Design Spaces (ODS 2010)
(Vol. 7, Iss. 2, pp. 23-35)

SourceBinder: Community-based Visual and Physical Prototyping

Bettina Conradi°, Balázs Serényi*, Miriam Kranz°, Heinrich Hussmann°
University of Munich°, Visual MINDS*
{bettina.conradi, hussmann}@ifi.lmu.de, art@visualminds.hu, kranzm@cip.ifi.lmu.de

Abstract. Physical interfaces broaden the entrance into the virtual world through sensors and actuators in our surrounding. Prototyping these interfaces demands expertise in hardware and software development – skills that are rarely found in end users, hobbyists or designers. If those users want to build a rapid prototype for a quick exploration of an idea, they are often troubled with learning the necessary programming and hardware engineering skills. The entry barriers for these target users can be lowered by providing suitable hardware and software toolkits. SourceBinder is a web-based visual programming tool that enables users to create projects and share them in a community. Users can test and adapt existing projects and even become developers by creating new nodes that can be used by the community. We want to present our extensions of SourceBinder which enables hardware to be connected to the visual programming environment, and show some example projects that can be realized in such a setting.

Problem statement & motivation

> In biological systems, there is a tendency for specialised organisms to win out over generalised ones. My argument is that the evolution of technology will likely be no different. Rather than *converging* towards ever more complex multifunction tools, my claim is that going forward we must *diverge* towards a set of simpler more specialised tools. (Buxton, 2001)

Although Buxton's claim about the need for more specialized devices is highly controversial, actively promoted research domains like tangible or ambient interfaces and successes in industry (e.g Nintendo Wii input devices, Guitar hero

for game consoles, the customizable Nabaztag rabbit[1]) show the interest of users in easy-to-use one-purpose tools. Tangible interfaces allow us to experience and manipulate digital information with our hands and sense of touch through specialized devices (Ishii & Ullmer, 1997) (Holmquist, Schmidt, & Ullmer, 2004). Ambient interfaces present information in the periphery of our perception through everyday artifacts in our daily life (Weiser & Brown, 1996) (Gross, 2003). As we are moving from a GUI-based interaction with the computer towards a more natural interaction that involves everyday objects, our movement, gestures and senses, combining suitable input and output modalities without converging too much functionality in one interface is key to success and adoption of an interface.

Giving end users the ability to participate actively in the development of applications proved to be successful for the Web 2.0, where users are moving from a consumer to a producer role (Fischer, 2009). For physical interfaces some people also try to refit them for their needs as can be seen in MAKE magazine and various other DIY-magazines and workshops (e.g. dorkbot[2], Makerfaire[3]). A study carried out by Hartmann et al. showed that product designers as well as hobbyists developed different strategies to *glue* together existing hardware and software components to create *ubicomp mash-ups* (Hartmann, Doorley, & Klemmer, 2008). Although interfaces to hardware and software are often well defined, programming effort is mostly inevitable in order to glue together components. Some users apparently are motivated to learn the needed skills to adjust physical interfaces and create code to connect software and hardware components, others do not feel skilled and proficient enough to hack and mash assembled devices and program script code. How can we make use of this continuum of participation? How does a prototyping environment for creating physical interfaces need to be designed in order to integrate users at different levels of participation?

After presenting recent advances in hardware and software prototyping toolkits for creating physical interfaces, we introduce SourceBinder, a web application that enables users to visually bind together hardware and software components and let them actively participate in the development and adaption of new components. After describing the general concept and our extensions for using the Arduino hardware, a short example illustrates the usage of SourceBinder and Arduino. The strategies we considered for the community-based development of SourceBinder are summarized and first observations and a short evaluation is explained.

---

1 http://www.nabaztag.com

2 http://www.dorkbot.org/

3 http://makerfaire.com/

Related work

A physical interface consists both of several hardware and software components that need to be connected/bound/glued together. After presenting related work in these separate domains, we want to introduce toolkits that already try to combine hardware and software prototyping.

Sketching in hardware for non-technical users like artists, hobbyists or even children is getting easier with toolkits like Phidgets (Greenberg & Fitchett, 2001), Smart Its (Gellersen, Kortuem, Schmidt, & Beigl, 2004), Calder Toolkit (Lee et al., 2004), LittleBits[4], Lego Mindstorms[5], Arduino[6], Electronic Bricks[7] and many more (see (Cottam & Wray, 2009) or (Moussette, 2007) for an overview). Assembling hardware becomes nearly as simple as plug and play. Arduino for example is used in classrooms to teach basic skills in electronic and to lower the barrier for tinkering with hardware, but also academia is using Arduino for building interface mockups. The active user community around Arduino has already created a lot of additional tutorials and software plugins which make the toolkit even mightier. The great interest of non-expert users in implementing their own hardware prototype is demonstrated heavily on Flickr and YouTube with a total of more than 50'000 uploads tagged with "Arduino". Also conferences like "Sketching in Hardware"[8] and workshops like "DIY for CHI: methods, communities, and values of reuse and customization" (Buechley, Rosner, Paulos, & Williams, 2009) promote advances in this area.

Lowering the barriers for programming software is also an active research domain. Possibilities are manifold, e.g. visual programming, tangible programming or animation software (Kahn, 1996). Visual programming hides the underlying complex code with graphical symbols that can be reused and combined with other blocks. Alice is used to introduce students to programming and lets them build their own 3D animations (Cooper, Dann, & Pausch, 2000). Agentsheets lets users build their own simulations in their domain of interest (Repenning, 1993). Microsoft's Kodu[9] enables children to create their own game by defining simple rules with input- and output-events, that can be shared in the XBox community. Max/MSP[10] is a commercially sold product that allows music artists to program audio signal processing code with graphical objects.

---

4 http://littlebits.cc

5 http://mindstorms.lego.com

6 http://www.arduino.cc/

7 http://www.seeedstudio.com/depot/electronic-brick-c-48.html

8 http://sketching10.com/

9 http://research.microsoft.com/en-us/projects/kodu/

10 http://cycling74.com/products/maxmspjitter/

Visually programming software and hardware is more difficult, since the physical hardware prototype has to be connected to the software workbench and changes in one part have to be reflected to the other. d.Tools allows prototyping with the Arduino hardware by letting users model the interface and state transitions visually (Hartmann et al., 2006). It also integrates a test and analysis mode for inspection of logged user tests. Scratch is originally developed for children to create multimedia applications (Resnick et al., 2009), but was already extended by an active user to allow Arduino to communicate with Scratch[11]. Other examples are the Lego Mindstorms NXT software (based on LabView), Fischertechnik's ROBO PRO[12] or Physical Etoys[13]. While these environments allow end-users with little programming knowledge to create complex physical interfaces, and people with more knowledge can often extend them with little scripts, these extensions mostly stay in the hands of the experienced users and the community cannot profit from them.

With SourceBinder we want to broaden the ecology of participation in an online visual programming environment, where components can be created, shared, rated and adapted in the community.

SourceBinder

SourceBinder[14] is a web-based tool in Beta status for visually creating Flash applications by binding nodes together (see Figure 1) that enables users to become active developers. *Nodes*, the building blocks of SourceBinder are regular ActionScript classes. They can be classes providing simple functionality but they also can be complex components as well. SourceBinder comes with most of the intern Flash functions built in as regular nodes, and some elements of favorite open source packages like Papervision 3D library, the WOW physics library, WiiFlash package to handle WiiMote, the as3glue package for dealing with physical computing and nodes providing common webservices like YouTube, Yahoo, and Google Maps. To allow engagement of users on different levels of complexity (Fischer, 2009) a node has three states, each allowing more possibilities to change the behavior. 1) In its basic state a node is just depicted by a symbol and allows connecting inputs and outputs of the node. 2) In the second state, the public attributes of the node become visible and can be changed directly or through binding it to another node (see Figure 2). 3) The third state shows the source code that can be changed and compiled. Thus a node can be adapted according to the user's knowledge level, either by just binding it to other nodes,

11 http://scratchconnections.wik.is/index.php?title=User:Chalkmarrow/Catenary

12 http://www.fischertechnik.com

13 http://tecnodacta.com.ar/gira/projects/physical-etoys/

14 http://sourcebinder.org/

by further influencing attributes or by directly changing the code. Therefore SourceBinder can be seen as a repository for community-built software components (Wulf, Pipek, & Won, 2008) with a graphical interface to combine, modify and extend them. Nodes can be combined to a network of nodes which produces a program which is called a *composition* (see Figure 1).

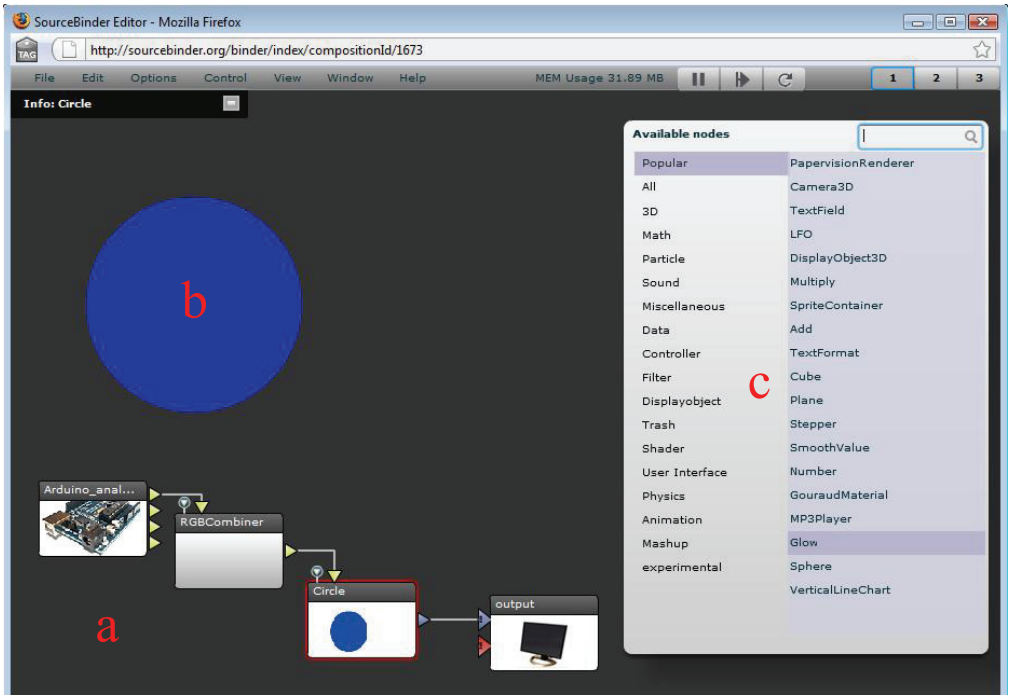

Figure 1: Composition interface: (a) Nodes are bound together, (b) a preview of the composition is shown in the background, (c) new nodes can be added to the composition via a browsable menu. In this example an analog sensor controls the blue pigment content of the circle

Figure 2: Attribute inspector of one node (second state of a node). (a) Attributes can be modified directly, (b) be bound to the output of another node or (c) be published to the global attributes panel for direct modification within the whole composition. (d) The source editor (third state of a node) is accessible at the bottom of the node.

Arduino extension

In order to test the extensibility of SourceBinder and building a platform for physical prototyping, we extended SourceBinder with Arduino nodes (see Figure 3). An Arduino Duemilanove board consists of a microcontroller and several pins for analog and digital reading and writing of attached sensors and actuators. We chose to implement one node for each functionality (analog r/w, digital r/w) to not overburden one node. The first is for analog reading of signals (e.g. sensors for light, temperature, pressure). If a sensor is attached to one of the analog pins, its value can be read by binding one of the outgoing arrows that correspond to the pin to another node. The second one is for digital writing (e.g. LEDs) and receives values over the incoming arrows that represent the pins on the Arduino board. The third experimental node is for analog writing (e.g. motors, RGB LEDs) and receives values from 0 to 1023 over the incoming arrows.

Figure 3: From left to right: (1) node for retrieving analog sensor input from the Arduino Duemilanove board, (2) node for writing digital values, (3) experimental node for writing analog values

The basic procedure for connecting an Arduino Duemilanove board is as follows: In order to let web-based Flash applications communicate with your computer a special policy file has to be adapted and run. This complication is due to the browser based design of SourceBinder and would not be needed if it was a standalone application. In addition, the Arduino needs to have Firmata (a special firmware) (Steiner, 2009) uploaded in order to communicate with SourceBinder. This connection process is not the easiest for novice users and needs to be further simplified.

Physical mood interface: demonstrating SourceBinder in use

In order to show the basic process for creating a project in SourceBinder, we want to illustrate the creation process of AngryBall (see Figure 5). It lets users share their angry emotions via Twitter by punching, hitting or pressing a rubber ball. This project was composed by a student who implemented a node for sending Twitter messages. For clarity, we only explain the binding process of the finished nodes not the implementation details.

Jan comes back home after an exam that didn't work out well. He wants to inform his Twitter friends about his feelings but is way too stressed and angry at

the moment. Punching a ball to let loose his emotions he thinks of a new kind of interface where he can send Twitter messages simply by squeezing his “angry ball”. A small pressure sensor fits perfectly into the ball and he connects the pressure sensor to his Arduino Duemilanove board. He opens up SourceBinder and drags the *arduino_analog_read node* on the composition area (see Figure 4). This node has several outgoing arrows that transport analog values (0-1023), each symbolizing one analog pin on the Arduino board. He connects one outgoing arrow of the node to the *GreaterThan node*. If the pressure sensor reaches a certain value it should trigger the Twitter message. He modifies the value of the compare node in the attribute inspector to 500 and connects its output to the SendTwitterMessage node. He adjusts the attributes of this node to his username, password and the message he wants to send (“I am sooo angry”). As soon as the Arduino node receives a value greater than 500 from the pressure sensor, the comparison node evaluates the result as true and triggers the SendTwitterMessage node that directly posts the message on the Twitter website.

Figure 4: Composition of the AngryBall project

Further projects with Arduino

Several projects have been created at our lab using SourceBinder and its Arduino extensions. Since new web services needed to be integrated for these projects (e.g. sending a Twitter message, opening a Skype message window) new nodes had to be created with the source editor.

The *Weather Station* retrieves information from the Yahoo Weather Channel and presents this information in an ambient interface prototype made of cardboard (see Figure 5). It uses the *Yahoo Weather Channel node* that retrieves weather information to a given location code. The location code is inserted in the attribute viewer of the *Yahoo Weather Channel node* and returns the temperature, a condition text (e.g. rainy, stormy etc.) and a title. The condition text is then evaluated with the c*ontains node*, if it matches a specified text (e.g. rain). If the weather forecast contains the word rain, the Boolean value “true” is send to one input of the *arduino_digital_write node*. This node has one entry arrow for each of the analog pins on the Arduino Duemilanove board. In our case, three LEDs for each weather condition are connected to the analog pins that reside in a cardbox. A LED is activated as soon as the according weather condition is evaluated positively.

In *FriendWatcher* little dolls represent friends in Skype[15]. Their noses display presence information and are shining green as soon as the person is online. If the doll is pressed, a conversation is started immediately. This composition contains two Arduino nodes. One for activating the LED (writing a digital value) and one for reading values from the pressure sensor (reading of analog values). In this case a student built a *Skype status node* that returns a boolean value if a specified user is online. This node is connected to the *arduino_digital_write node* to activate the LED. The other node is the *FriendsWatcher node* that opens a Skype message box when it receives input. This node is connected to the *arduino_analog_read node* and is activated as soon as the pressure sensor is pressed.

All of the mentioned projects use web services to retrieve or manipulate digital information in the World Wide Web. Besides these also other projects are possible, e.g. to manipulate visualizations or audio with physical handles.

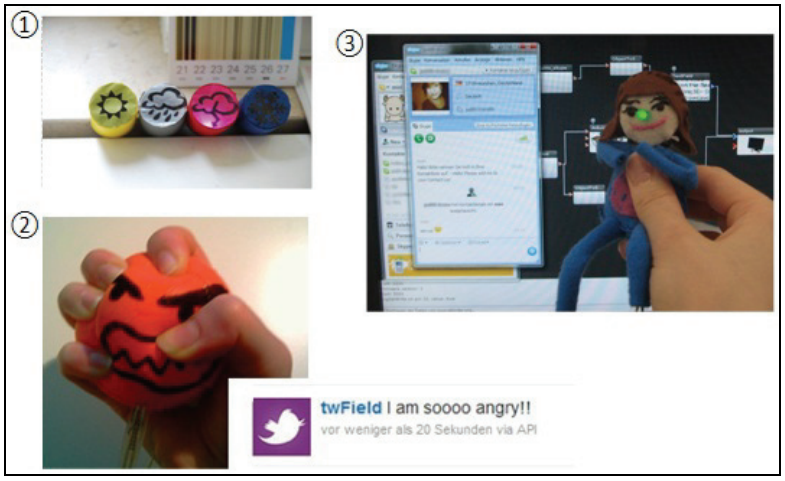

Figure 5: Exemplary Projects: 1) Weather Station 2) AngryBall 3) FriendWatcher

Community features

SourceBinder is thought of as a prototyping platform that evolves with the help of an active community. Therefore strategies for building active and engaged online communities are implemented as follows:

Seeding, Evolutionary Growth and Reseeding (Fischer, 1996)*:* An active community has to be provided with *seeds* (artifacts that engage people in using the application and allow for modification and extension by the community). These seeds serve as a starting point for the evolutionary growth phase where users can extend and modify the current version. The available nodes can serve as *seeds* that allow users to create first compositions. As our first exemplary projects

[15] Skype, an instant messaging and video conferencing service, http://www.skype.com

showed these nodes are sufficient for projects that base on current compositions, but not every imagined project could be created with only using existing nodes. The evolutionary grow phase will show if enough active users with programming skills are motivated to contribute nodes to the community and enriching the library of available nodes.

Clear Authorship and Use License Attribution Systems (Monge, Ovelar, & Azpeitia, 2008): Each node only has one author. The author can determine if he wants his node to be either *private* or *public*. Thus users can create first test nodes in a private environment and decide later to contribute it to the community. If the status of one node is set to public, it is immediately visible in the node library of the other users. Others are now free to use it in their compositions. If they want to modify the source code they can *fork* it (build a copy of it) and customize it in any way they want and now possess authorship over this new node. This can result in a nice iteration, where an idea evolves through the help of other users.

Rapid Content Creation Systems (Monge, Ovelar, & Azpeitia, 2008): To encourage and engage new users SourceBinder offers "Getting Started" compositions to learn the basics about "binding". Recent, popular and featured compositions are presented on the website to get inspired by other projects. Projects can be opened, edited and saved as own compositions. Users can also join a interest group to discuss, share and collaboratively tailor nodes for a specific application area. These groups can also help interested beginners in getting into the details of SourceBinder and creating own nodes.

Reputation Systems for Contents (Monge, Ovelar, & Azpeitia, 2008): To motivate users and integrate feedback mechanisms, a user can see how many views and comments he got from other users (see Figure 6) and can in turn vote for his most loved projects. *Binderpoints* give a user feedback about his binding activity.

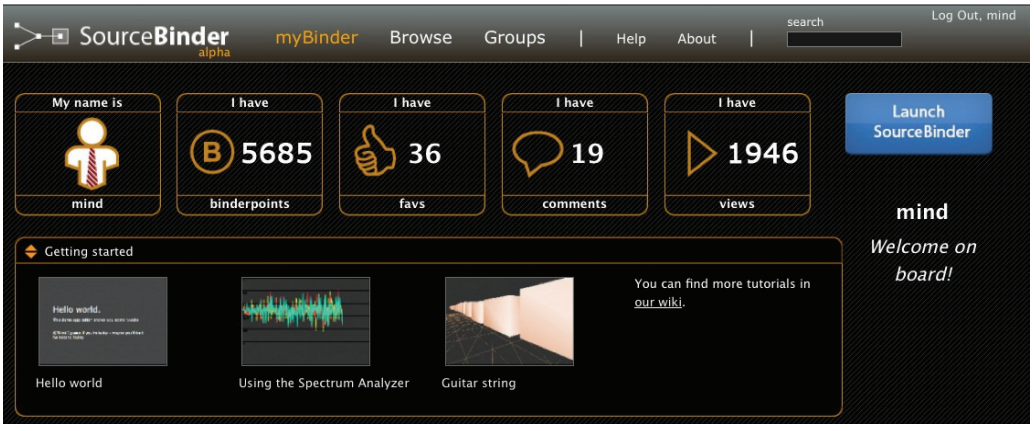

Figure 6: Personal view of one user with feedback mechanisms (binderpoints, favorites, comments, views)

In summary, SourceBinder allows users to engage on different participation levels. Users can either only browse existing projects, they can slightly modify them or build new compositions with available nodes. They can program their own private nodes and compositions, but they can also contribute them to the community and get actively involved in an interest group.

Evaluation

Before opening SourceBinder to the broad public, community-specific pitfalls in the current design have to be identified. We chose to assess SourceBinder with the cognitive dimensions framework for visual programming environments (Green & Petre, 1996), because it allows a broad-brush evaluation with a framework for the most important usability flaws to consider for visual programming. We want to discuss two that are especially critical in community-based development:

Secondary notation includes techniques that convey extra meaning to the reader, e.g. comments, colors, layout. In a community it is very critical to have a precise description of components and whole applications. Users need to have the possibility to annotate their own projects for their own use but also for other users to understand the composition and single nodes. SourceBinder offers the possibility to add descriptions to a node and comment whole projects. What is missing is a commenting function in the projects to annotate groups of nodes and connections between nodes, otherwise projects cannot be adapted easily. Also users should be able to tag nodes to gain more metadata about the node repository (Monge, Ovelar, & Azpeitia, 2008) and allow additional ways to search the node library instead of the proposed categories.

Consistency is another major problem in community-based development. As soon as users start developing their own nodes, they are determining the name, the public visible attributes and the underlying functionality. Some nodes may contain only one single functionality; others may contain a small composition in itself. Nodes can be created that have slightly the same functionality and are also named somehow similar (e.g. three different nodes for sinus calculation: sin, Sin, sinnn). This also affects the findability of nodes: If nodes are not named properly or contain too much invisible functionality, other users cannot adapt that node and will create their own nodes. This can lead to a vast amount of nodes that have slightly similar function. With a growing node set it will be problematic to stay consistent. Possible solutions can be clear guidelines for naming conventions and structure of a node or an approval process of new nodes before making them public. Rating functions could be integrated into the decision by relying on the most appreciated nodes. Active users could be assigned an administrator status to check new nodes for their consistency.

Further usability issues were found during a student's project with SourceBinder and concern the extension of the visual programming environment with hardware. An urging problem is that it is often difficult for non-experts to *connect the hardware* to the software workbench. In this case they have to load firmware on the microcontroller, enable security settings and start a proxy - a task that frightens novice users and might even scare them away. Another problem is that *changes in the hardware are not clearly visible* in the software environment. Although text nodes can be bound to the output of the Arduino node to visualize values, changes would be better visualized automatically in the preview area of the composition.

In near future we want to further heuristically evaluate SourceBinder and afterwards test it with students in a one week workshop to find out about usability issues concerning the visual programming part. As soon as SourceBinder is opened to the public we will analyze the phase of evolutionary growth, e.g. user commitment (roles, usage behavior etc.), quality and the overall structure of created components. This will be achieved with observations and interviews with active users. The insights gained from these observations will then affect the reseeding phase.

Conclusions & future work

Our initial evaluation and explorations of example projects with SourceBinder for visually programming physical prototypes assured us in further investigating its potential benefits. Non experienced people in hardware or software engineering are enabled to "bind" together application logic for creating experience prototypes. They can rebuild projects of other users and can contribute to the mightiness of the toolkit by composing new nodes. Thus SourceBinder can serve as a repository for user generated components. Problems with a community-based component repository development (e.g. consistency of developed nodes) need to be carefully watched in the initial phase after public release of SourceBinder. As SourceBinder is not public yet, we are still able to enrich the seeds sawn for new users, thus we want to discuss possible evaluation and seeding strategies for the community development of SourceBinder at the "Open Design Spaces" workshop.

Future work will include better presentation possibilities of physical prototypes in SourceBinder. Besides the textual description, pictures or videos of the prototype in action need to be supplied. The hardware setup with sensors, resistors etc. needs to be accessible in order for others to rebuild it. Fritzing lets users graphically model a microcontroller like Arduino that is connected to sensors and actuators via a breadboard (Knörig, Wettach, & Cohen, 2009). An integration of Fritzing with its visualization of the involved hardware could greatly benefit the shareability of projects.

References

Buechley, L., Rosner, D. K., Paulos, E., & Williams, A. (2009): 'DIY for CHI: methods, communities, and values of reuse and customization', *Conference on Human Factors in Computing Systems*, pp. 4823-4826.

Buxton, W. (2001): 'Less is more (More or less)', In *In The Invisible Future: The Seamless Integration of Technology in Everyday* (pp. 145-179), New York: McGraw Hill.

Cooper, S., Dann, W., & Pausch, R. (2000): 'Alice: a 3-D tool for introductory programming concepts', *Consortium for Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107-116.

Cottam, M., & Wray, K. (2009): 'Sketching Tangible Interfaces: Creating an Electronic Palette for the Design Community', *IEEE Computer Graphics and Applications*, vol. 29, no. 3, pp. 90 - 95.

Fischer, G. (1996): *Seeding, Evolutionary Growth, and Reseeding: Constructing, Capturing, and Evolving Knowledge in DomainOriented Design Environments* (pp. 135-143), Malmö University, Sweden.

Fischer, G. (2009): *End-User Development and Meta-design: Foundations for Cultures of Participation*, (V. Pipek, M. B. Rosson, B. Ruyter, & V. Wulf)*End-User Development*, Lecture Notes in Computer Science (Vol. 5435, pp. 3-14), Berlin, Heidelberg: Springer Berlin Heidelberg.

Gellersen, H., Kortuem, G., Schmidt, A., & Beigl, M. (2004): 'Physical Prototyping with Smart Its', *IEEE Pervasive Computing*, vol. 03, no. 03, pp. 74-82, Published by the IEEE Computer Society.

Green, T., & Petre, M. (1996): 'Usability Analysis of Visual Programming Environments: A Cognitive Dimensions' Framework', *Journal of Visual Languages and Computing*, pp. 131 - 174.

Greenberg, S., & Fitchett, C. (2001): 'Phidgets: easy development of physical interfaces through physical widgets', *Symposium on User Interface Software and Technology*, pp. 209 - 218.

Gross, T. (2003): 'Ambient interfaces: design challenges and recommendations', *Human computer interaction: theory and practice*, pp. 68-72.

Hartmann, B., Doorley, S., & Klemmer, S. R. (2008): 'Hacking, Mashing, Gluing: Understanding Opportunistic Design', *IEEE Pervasive Computing*, vol. 7, no. 3, pp. 46-54, IEEE Computer Society.

Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., & Gee, J. (2006): 'Reflective physical prototyping through integrated design, test, and analysis', *Symposium on User Interface Software and Technology*, pp. 299 - 308.

Holmquist, L., Schmidt, A., & Ullmer, B. (2004): 'Tangible interfaces in perspective', *Personal and Ubiquitous Computing*, vol. 8, no. 5, pp. 291-293.

Ishii, H., & Ullmer, B. (1997): 'Tangible bits: towards seamless interfaces between people, bits and atoms', *Conference on Human Factors in Computing Systems*, pp. 234 - 241.

Kahn, K. (1996): 'Drawings on napkins, video-game animation, and other ways to program computers', *Communications of the ACM*, vol. 39, no. 8, pp. 49 - 59.

Knörig, A., Wettach, R., & Cohen, J. (2009): 'Fritzing: a tool for advancing electronic prototyping for designers', In *Proc. TEI'09* (pp. 351 - 358).

Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., & Leigh, D. (2004): 'The calder toolkit: wired and wireless components for rapidly prototyping interactive devices', *Designing Interactive Systems*, pp. 167-175.

Monge, S., Ovelar, R., & Azpeitia, I. (2008): 'Repository 2. 0: Social Dynamics to Support Community Building in Learning Object Repositories', *Interdisciplinary Journal of of E-Learning and Learning Objects* , vol. 4.

Moussette, C. (2007): 'Tangible interaction toolkits for designers', *Scandinavian Student Interaction Design Conference*.

Repenning, A. (1993): 'Agentsheets: a tool for building domain-oriented visual programming environments', *Conference on Human Factors in Computing Systems*, pp. 142-143.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., et al. (2009): 'Scratch: programming for all', *Communications of the ACM*, vol. 52, no. 11, pp. 60 - 67.

Steiner, H. (2009): 'Firmata: Towards making microcontrollers act like extensions of the computer', *New Interfaces for Musical Expression*, pp. 125-130.

Weiser, M., & Brown, J. S. (1996): 'Designing calm technology', *PowerGrid Journal*, vol. 1.

Wulf, V., Pipek, V., & Won, M. (2008): 'Component-based tailorability: Enabling highly flexible software applications', *International Journal of Human-Computer Studies*, vol. 66, no. 1, pp. 1-22.