Multiple-perspective interfaces for software development environments

Agnes Chang

Thesis Proposal for the Degree of Master of Science at the
Masachusetts Institute of Technology
Fall 2009

Thesis Advisor David Small
Associate Professor
MIT Media Laboratory

Thesis Reader Mitchel Resnick
LEGO Papert Professor
of Learning Research
MIT Media Laboratory

Thesis Reader Casey Reas
Professor
UCLA Design | Media Arts

Table of Contents

1. Abstract

One of the major barriers designers and artists encounter when programming digital media is the difficulty of translating the mental models of their creations into a format and language that can be interpreted by computers. Creative people exhibit a variety of ways of thinking, and the constant necessity to translate between their personal mental model and the programming paradigms dictacted by current software representations limits the programmer's creative potential.

This thesis proposes to research, design, and implement a novel interface that enables the programmer to define a conceptual visual representation of computation to complement the traditional text-based code. In particular, this work will focus on the following: the design of the set of visual vocabulary necessary for idiosyncratic conceptual representation, the maintenance and display of relationships between the visual and textual elements, and finally interface support of multiple programming styles throughout different programming stages of writing, reading, and debugging. The thesis seeks to offer alternative methods of organizing, understanding, and learning programming so that software can be a more accessible and expressive medium for all types of designers and artists.

2. Motivation

Artists and designers often have clear intentions of what they would like to create in the software medium, as well as mental models of how to go about achieving the desired output. The real difficulties are encountered when translating the mental models of their creations into code that can be interpreted by computers. This thesis seeks to alleviate the tedious and redundant aspects of congitive translation that currently form a significant bottleneck in the software creative process.

In the creative process (Fig. 1), the artist begins with an intention (be it well-formed or indefinite), then forms an internal model about the problem, and translates the model into code, which in turn produces the output program. The translation process is difficult due to incongruities between individuals' approach to problems and the design of the particular programming language. This step poses an obstacle even for expert programmers: after setting a project aside for a short amount of time, when coming back to the code, even the original author usually has difficulties recalling their train of thought when they were writing the program.

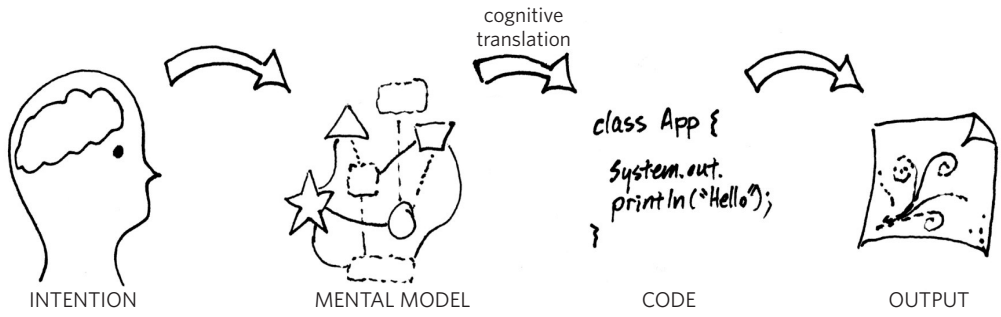
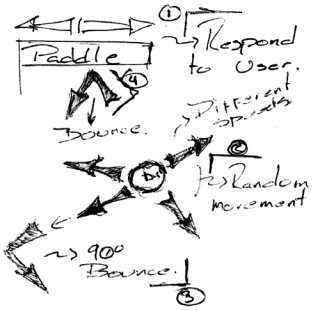
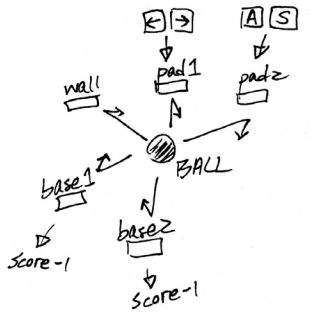


Fig. 1: The creative process.

More importantly, people exhibit a variety of approaches to thinking about problems [16]. A preliminary examination of individuals' visual representations of their mental models (Fig. 2 through 4) demonstrate that mental models comprise a variety of information. Some are functional descriptions, such as "bounce a ball" (Fig. 2), while others are system elements, such as "interface" and "buttons" (Fig. 4). DiSessa [2] has identified these types of information as belonging to a functional model versus a structural model, respectively. Each individual's mental model is a combination of mental and structural models fromed from personal experience — experience with various programming paradigms, experience with the platform of choice, and experience with the nature of the project they are trying to create, etc.





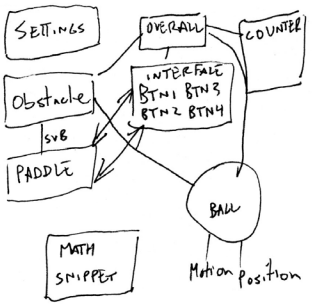


Fig. 2 to 4: Examples of three individuals' conceptualization of the classic two-player pong game.

However, a multiple perspective interface that enables documentation of the mental model in complement with traditional text-based software creation can ease many of the difficulties with cognitive translation. It is the intention of this thesis to demonstrate that by allowing users to represent their idiosyncratic mental models and to use such representations to create, maniputlate, and explore their programs, the process of creative coding can become more direct and intuitive.

Further, since the nature of creative work is reiterative, throughout this process the mental model is continuously revised and cognitive translation is continuously revisited. The creative process also applies at different scales: in the pong scenario, the author might have intentions and models for the behavior of a singular ball as well as a vision of a meta set of pong variations, and there are cognitive translations necessary at each of these scales.

3. Context

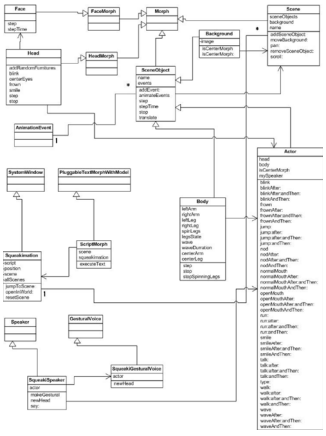


Fig. 5: The Unified Modeling Language has a strict ruleset governing its visual vocabulary and syntax. From [17].

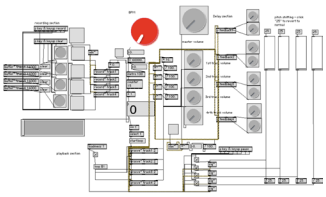


Fig. 6: Screenshot of the Max/MSP visual language, which employs a "river metaphor" for programming streaming audiovisuals. From [8].

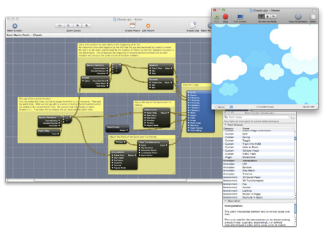


Fig. 7: Quartz Composer is a visual programming language based on the depiction of dataflow. From [12].

While no work has yet directly addressed the cognitive translation gap, this thesis is informed by long traditions of research in software writing and software comprehension tools. In particular, the ideas proposed in this work are a derivative of the inter-related fields of software visualization, graphical modeling languages, and visual programming languages, each of which have sought to address various bottlenecks in the creative process.

Much of software comprehension tools focuses on illustrating the hidden mechanisms of the software, i.e. the structural model, such as data flow and control flow. Software visualization designs vary from the aesthetic [7] to the analytic [3], and some are interactive [9]. Graphical modeling languages, such as the industry standard Unified Modeling Language [17], aimed to develop a standard visual language to describe system structures. However, in addition to passing over functional information, software visualization and graphical modeling languages also operate disjoined from the programming activity itself.

Certain visual programming languages employ structural perspectives which offer alternative programming paradigms that align more closely with common mental models for the target tasks these environments were designed to solve. Languages based on dataflow, such as Max/MSP [8], vvvv[18], and Quartz Composer[12] aptly employ the metaphor of river and tributaries for projects based on streaming audio and video. However, these metaphors are not easily generalizable to other tasks.

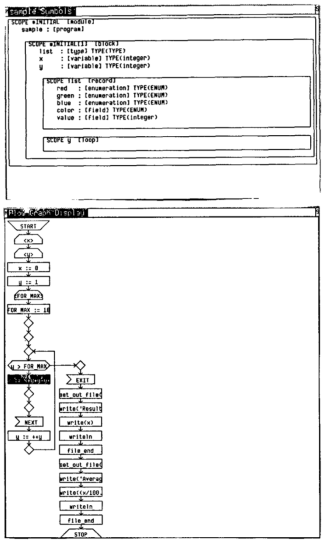


Fig. 8 & 9: Screenshots of PECAN (1984), a program that supported toggling between multiple views of stored data structures. From [14].

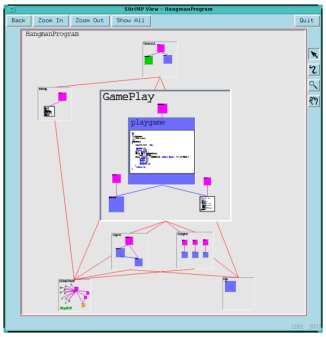


Fig. 10: Screenshot of SHriMP (1999), designed to support the construction of a mental model during software exploration. From [15].

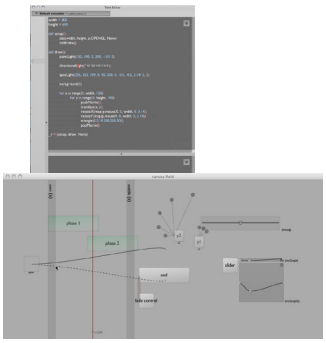


Fig. 11: Screenshot of Field environment, which enables visual combination of code, interface, and output elements. From [4].

Several research studies have also looked into the concept of a software tool that presented multiple views. An early example is PECAN [14], a program environment that supported toggling between multiple views of stored data structures, although the multiple views did not apply to the program structure itself. The SHriMP tool [15] specifically addresses a variety of cognitive models in its design, but as an exploration tool, it was not designed to accommodate the writing and editing of code.

Of recent work, the Field programming environment [4] still in the beta development phase also takes a multi-paradigm approach, and more specifically, the complement of code with user-defined visual abstractions of code. In addition, the Field environment enables the combination of user-defined visual elements, with GUI elements that can replace specific code syntax, with elements of visual output. Unfortunately, the combination on a single canvas of so many visual elements that are semantically distinct requires even more translation and causes cognitive overload.

4. Methodology

This thesis proposes to address the cognitive translation gap by researching, designing, and implementing a multiple-perspective interface that allows programmers to manipulate their program via a visual representation of their mental model. Fig. 2 is a preliminary sketch of the interface, illustrating an example of a pong program. On the left side the programmer can create their own mental representation, and on the right is traditional code representation. The visual elements on the left will be linked to their respective code fragments, so that upon selection of a visual element on the left, the text display on the right automatically navigates to the associated code.

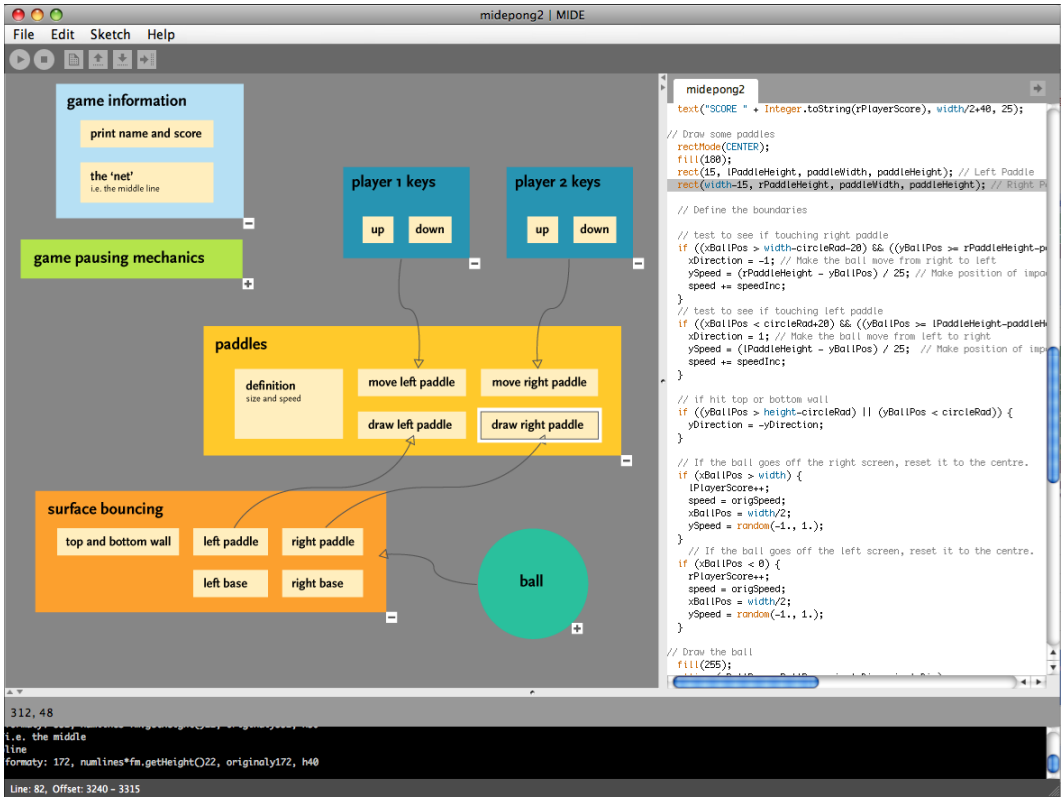


Fig. 12: Screenshot of the complementary visual-textual interface. By selecting the "draw right paddle" visual element, the text representation automatically navigates to the associated code. The code here is a Processing implementation of the classic two-player pong game, cf. Fig. 3.

The goals of this interface are threefold. First, to allow programmers to organize their code along visual and spatial dimensions, such as color, shape, and location. Secondly, to serve as an extension of memory for users, so that they can concentrate their efforts on the design of the project without needing to continuously undergo the same mental translations as they reiterate over they designs. Finally, such an interface can aid the learning process by allowing users to reference their own past work, as well as examine other people's thought processes and approaches in similar scenarios.

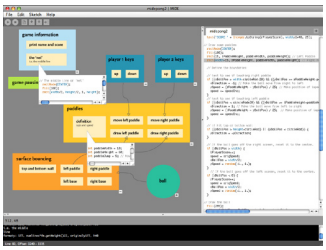


Fig. 13: Sketch of non-linear code access via the visual representation.

At the same time, the approach of this thesis is limited by the choice of addressing of idiosyncratic mental models. Because mental models differs per individual, the interface will not and cannot assume an exhaustive mapping from visual components to program structure. For the same reason, this interface will not attempt to generate code from the model, or a model from the code, unlike many software-writing tools today. Lastly, the scope of this thesis will focus on mental models, and other representations of code such as data flow, control flow, and event flow, which are all very important to our understanding of programs, will be considered in respect to mental models, but will not be addressed directly.

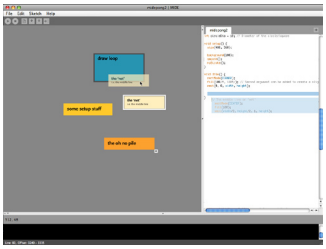


Fig. 14: Sketch of example of cross-representation correspondence: : drag-and-drop rearrangement of code.

To inform this design work, in addition to a literature review, a critical part of the initial stages of this thesis will be an ethnographic study aimed at understanding how users from the target demographic visually represent their mental models. The goals of the study will be to discover what visual elements people naturally use to depict their mental models, as well as, given a set of suggested visual elements, which are acquired for usage. Survey methods will be based upon one-on-one interviews as well as visual data similar to Fig. 1 through 3.

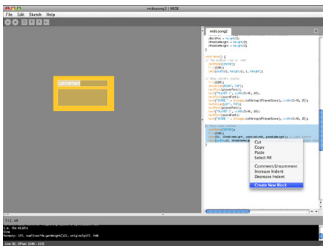


Fig. 15: Sketch of "bottom-up" programming approach: creating visual elements after text.

Finally, the implementation of this work will be a redevelopment of the author's previous multiple-perspective project called MIDE, and will use the same technical setup. The current system is primarily based on the Processing Integrated Development Environment (IDE) [11], available under the GNU General Public License, and the graph component of MIDE is currently implemented via a version of the JGraph library [6] that is available under the GNU Lesser General Public License. Processing is a programming language based on Java, designed for the electronic arts and visual design communities with the purpose of teaching the basics of computer programming in a visual context [13]. The decision to implement this work in the Processing IDE is based on the coincidence of purpose and on the consideration that members of the Processing community comprise the primary user base that this work intends to target.

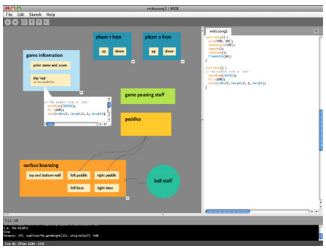


Fig. 16: Sketch of "top-down" programming approach: writing text after creating visual stubs.

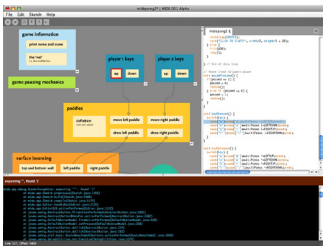


Fig. 17: Sketch of example of cross-representation correspondence: visual error notification.

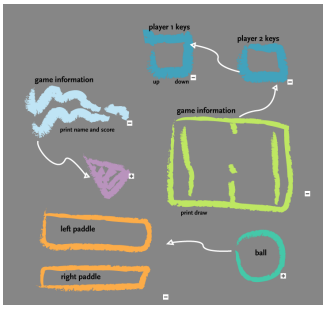
5. Contributions

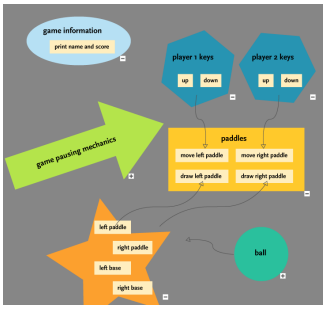
This thesis proposes to make its primary contributions in the following areas by answering some of these questions:

Expressive Graphical Tool: How broad a set of visual vocabulary, and what type of visual vocabulary is necessary for most people to express their mental picture? How might the design of 20% of possible visual vocabulary express 80% of users' needs? Is there a subgroup of needs for which this cognitive translation approach is particularly suited? What set of visual vocabulary is particularly suited for this subgroup?

Versatile Interface: How will the interface functionalities accommodate people's different programming styles and the different ways people might use their visual representations? For example, some users might prefer to write code prior to creating visuals, while others might try to create a visual outline before starting to fill in the code.

Flexible Visual-Textual Associations: How will the interface indicate cross-representation relationships to aid program comprehension? For example, a multiple-perspective interface could give visual indication of the parts of code that are throwing an error, versus the fragments that are compilable, and error codes could be printed in the color of the corresponding visual element. This design issue is of particular importance since the mapping from visual to textual is not exhaustive.





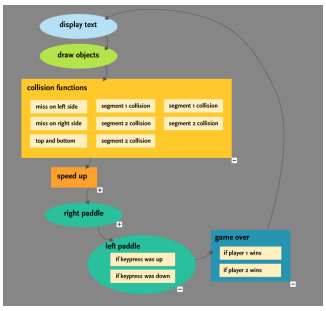


Fig. 18 to 20: Sketches of graphic possibilities for the visual representation.

6. Evaluation

The project is intended to be primarily a design work and will be evaluated in continual critiques by my advisor and readers. Through an iterative design process of research, planning, implementation, critique, and refinement, the work will be evaluated in light of the design principles identified and refined in the initial research.

The evaluation of the implementation will be conducted in two phases. The first, an informal alpha testing, will be mainly qualitative and observational, with a small number (<10) of subjects. The specific questions and goals of the alpha test will be determined by the findings of the ethnographic research to be conducted at the beginning of this project. Ideally the subjects of the alpha test will the same members as the initial survey, or of similar backgrounds.

The second phase is planned to be an external beta test over long-term use, probably one week of time, where members of the Processing community are invited to participate. Pre-and post-experiment questionnaires will be used to collect primarily qualitative data for the beta phase.

7. Thesis Readers

David Small is an assistant professor at the MIT Media Laboratory where he directs the Design Ecology Group. His work creates visual communication that incorporates new display and computational technologies, novel software techniques, and perceptual and cognitive issues. Small received his Ph.D. and M.S. in Media Arts and Sciences, and his B.S. in Cognitive Science, all from MIT.

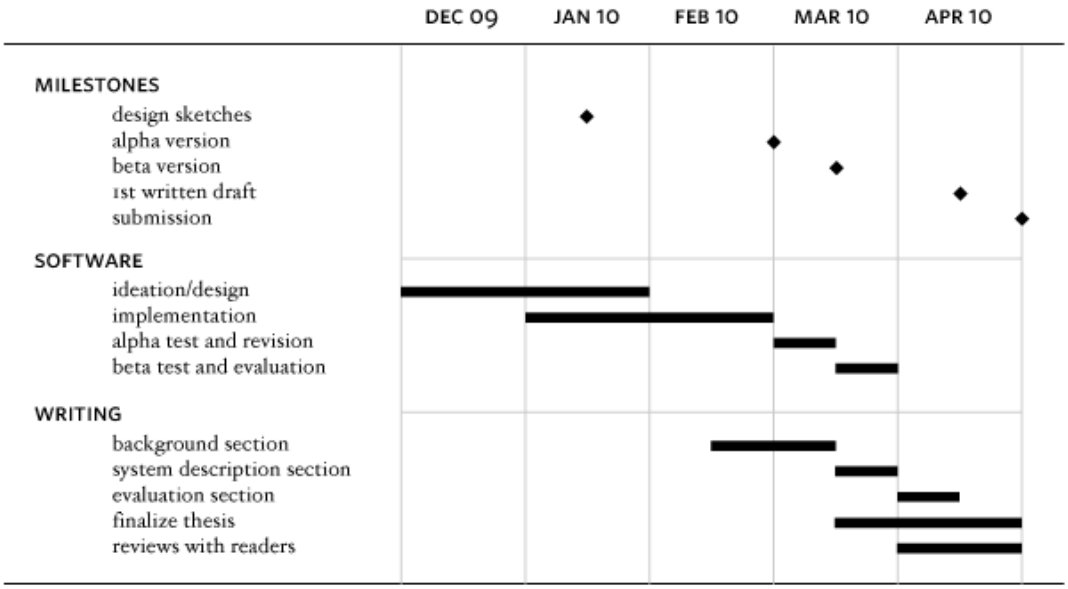
Mitchel Resnick, LEGO Papert Professor of Learning Research and head of the Lifelong Kindergarten group at the MIT Media Laboratory, explores how new technologies can engage people in creative learning experiences. Recently, Resnick's group developed Scratch, an online community where children use a graphical programming language to create and share interactive stories, games, and animations.

Casey Reas is a professor in the department of Design | Media Arts at the University of California, Los Angeles. He has exhibited his work internationally at institutions including Laboral (Gijon, Spain), The Cooper-Hewitt Museum (New York), and the National Museum for Art, Architecture, and Design (Oslo). With Ben Fry, he initiated Processing.org in 2001. Processing is an open source programming language and environment for creating images, animation, and interaction.

8. Resources

For the implementation of this software project, no addition resources are required beyond the computers and displays that currently belong to the Design Ecology Group.

9. Timeline



10. References

[1] M. M. Burnett and M.J. Baker, "A Classification System for Visual Programming Languages," Oregon State University Corvallis, OR, USA 1993.

[2] A. DiSessa. "Models of Computation." in Norman, D. A. and Draper, S. W. ed. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., 1986.

[3] S. Eick, J. Steffen, E. Summer. "Seesoft: a tool for visualizing line oriented software statistics." In Proc. of IEEE Transactions on Software Engineering, 1992.

[4] Field, by the OpenEndedGroup. http://openendedgroup.com/field/

[5] T. R. G. Green and M. Petre. "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework." Journal of Visual Languages and Computing, vol. 7. (1996) 131-174.

[6] JGraph. http://www.jgraph.com/

[7] E. Kleiberg, H. van de Wetering, and J. J. van Wijk. "Botantical visualization of huge hierarchies." In Proc. IEEE Symposium on Information Visualization, 2001.

[8] Max/MSP. http://www.cycling74.com/products/max5

[9] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. "Visualizing pro-

grams with Jeliot 3." In Proc. of the working conference on Advanced Visual Interfaces, 2004.

[10] W. B. Paley. "Code Profiles." http://artport.whitney.org/commissions/codedoc/Paley/CodeProfiles_800x600.htm

[11] Processing. http://processing.org/

[12] Quartz Composer, by Apple Computer, Inc. http://developer.apple.com/graphicsimaging/quartz/quartzcomposer.html

[13] C. Reas and B. Fry. "Processing: a programming handbook for visual designers and artists." 2007.

[14] S. Reiss. "PECAN: Program development systems that support multiple views." In Proc. of the 7th International Conference on Software Engineering, 1984.

[15] M. Storey, F. Fracchia, and H. Müller. "Cognitive design elements to support the construction of a mental model during software exploration." Journal of Systems and Software, 44. (1999) 171-185.

[16] S. Turkle and S. Papert. "Epistemological Pluralism: Styles and Voices within the Computer Culture." Signs, Vol. 16, No. 1. (1990) 128-157.

[17] Unified Modeling Language. http://www.uml.org/

[18] vvvv. http://vvvv.org/