Mediating Programming through Chat for the OLPC

Jill P. Dimond
Georgia Institute of Technology
85 5th St NW
Atlanta, GA 30332 USA
jpdimond@cc.gatech.edu

Sarita Yardi
Georgia Institute of Technology
85 5th St NW
Atlanta, GA USA 30332 USA
yardi@cc.gatech.edu

Mark Guzdial
Georgia Institute of Technology
85 5th St NW
Atlanta, GA 30332 USA
guzdial@cc.gatech.edu

CHI 2009, April 4 – 9, 2009, Boston, MA, USA
ACM 978-1-60558-247-4/09/04.

Abstract
We built a text-based programming environment that enables youth to design and implement a chat client for the One Laptop per Child XO. The environment allows users to program and chat simultaneously. We conducted two one-week workshops at a Girl Scout camp to test user engagement with the environment. In this paper, we examine how chat mediated the programming experience in a collocated environment and its implications for motivating participation in computing.

Keywords
Computer science education, programming, computer supported collaborative learning

ACM Classification Keywords
H5.m. Information interfaces and presentation, K.3.1 Computer Uses in Education

Introduction
In *Storm's Weekend with Rachel* [1], Bruckman tells the story of how a young girl signed up for MOOSE Crossing, a text-based programmable online virtual world, on a Friday night. When Bruckman returns on Sunday night, the girl, Storm, had created many complex computational artifacts with the help of a new friend Rachael whom she met within the virtual environment.

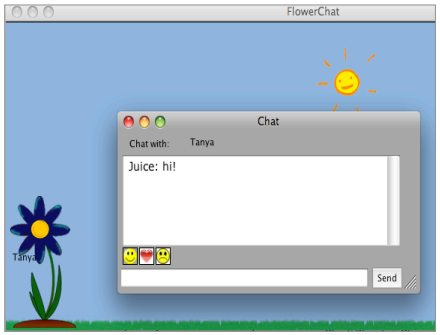
Ten years later, with an increased dearth of women in computing [11] and an overload of visually engaging interactive sites on the Web, could this happen again in a solely text-based environment? More importantly, can contextualizing programming in a rich social environment engage girls in programming activities? Recent approaches to elicit girls' interest in programming share Bruckman's strategy in terms of using storytelling as a motivational context, but use more sophisticated user interface designs. Examples include Storytelling Alice, Rapunsel, and Scratch [6,7,2]—all of which employ a graphical user interface with drag-and-drop programming statements. This type of interface has been used mainly to avoid compilation errors as well as to ground the application in contexts with which girls are familiar.

Our programming environment takes a different approach. We focused on a particular project—building a chat client for the One Laptop per Child XO—in a text-based programming environment. We leveraged two educational approaches: appealing to potential altruistic occupational desires that may be motivating for girls [4], and contextualizing them in activities which middle school girls are already engage in, social networking and chat [8]. Chat also has a collaborative affordance that can lead to a convergence of meaning and learning [10]. We gave them a simplified, functional chat client and instructed them to re-design the user interface for kids in developing countries. Our design enabled girls to chat with each other while programming. The girls were also co-located in a classroom environment, enabling them to socialize virtually or through speech. The programming language was also text-based. Our long-term goals involve exploring innovative approaches to broaden interest in computer science; however, in this paper, we concentrate on the effects of implementing a combined chat and programming environment and exploring use, behavior, and engagement among participants.

Our research question is: in what ways does chat embedded in the collocated environment mediate the programming experience? More specifically, 1) Do participants chat and program at the same time? 2) Do they give or receive programming help over chat? 3) How are the effects of being able to chat reflected in their end product code and computational artifacts? 4) What opportunities for learning, collaboration, and encouragement are enabled? We first describe the context of where our study was situated, then present the programming environment, and last describe results of the pilot study and discuss how our environment shaped the programming experience.

Location and Participants

The study was conducted over two weeks at a Girl Scout camp in the U.S. in the Summer of 2008. The workshop was broken into two sessions, five days each, and each participant was given her own PC laptop to use during the week-long session. The camp director sent out an email prior to the camp informing parents of the study. We received 17 responses prior to the beginning of the camp; 15 parents and children confirmed participation and turned in consent forms during check-in. Seven were African American and eight were Caucasian with ages ranging from 10-15. A researcher lived at the camp for the two-week period and interacted with participants informally outside of the study.



Screenshot of Flower chat with a chat window.

Name	Sent	Received
Allison	1216	665
Sara	452	313
Jennifer	301	791
Madison	196	215
Serena	176	175
Cassandra	165	107
Shelby	160	180
Sierra	152	152
Elizabeth	115	110
Miranda	113	127
Emily	105	111
Tanya	100	116
Heather	79	150
Kyra	73	91

table 1: Total number of messages sent and received by each participant

Programming Environment

Participants programmed on locally networked PC laptops and were able to see their completed programs on the OLPC. The programming environment allows a simple chat client's user interface (like MSN Messenger or Google GChat) to be redesigned with an embedded programming language. Programming commands and chat are entered into separate windows. In order to motivate participants, we asked them to improve the user interface for other children in different countries such as Haiti or Peru [5]. Influenced by Pane's HANDS event driven programming environment [9], we implemented a basic programming language with a simple debugger and limited complexity of the kinds of statements that could be made. Functions included changing chat properties such as the colors, images, sounds, and font as well as events that would trigger sound. The sequence of the programming statements also did not matter. Looping and advanced data structures were explicitly left out of the programming language.

An example of the code that changes a property is:

```
changeBackgroundImage "CloudyBG.jpg"
```

This code changes the current background image to the image "CloudyBG.jpg'' (Note that the command is not case sensitive.) An example of an event is:

```
if the chat text contains "lol" then
play "laugh.wav"
```

If the term "lol" is typed or received, the code instructs the program to play the "laugh.wav" file that plays a laughing sound. A library of images and sound files were included in the environment or participants could add their own images. Participants could modify three different chat interfaces: the standard chat list, Flower chat, and Circle chat. Participants could begin with either of these environments and customize them into a potentially more complicated layout/design. Participants were given a handout containing different "codes" associated with the chat layouts. We referred to the programming statements as codes because of the participants' familiarity with MySpace codes, and referred to the activity they were doing as programming.

Methods

We built a logger into the programming environment. The server-side logger captured all activity, including server connections, disconnects, and chat messages. It also logged the identity of the sender and receiver and which participant left a chat with another person. Group chat was not supported. The client-side logger polled every two minutes and captured participants' individual programs. The logger also captured when participants compiled their programs (by clicking "test" on the interface). Date/time stamps were logged for all activity on both server and client-side activity. Logs were loaded into MySQL and analyzed for temporal participation, interpersonal conversations, and chat and programming content.

Results and Discussion

Participants sent 4,252 messages to one another over the two weeks. Table 1 to the left shows the total number of messages sent and received by each

participant[1].

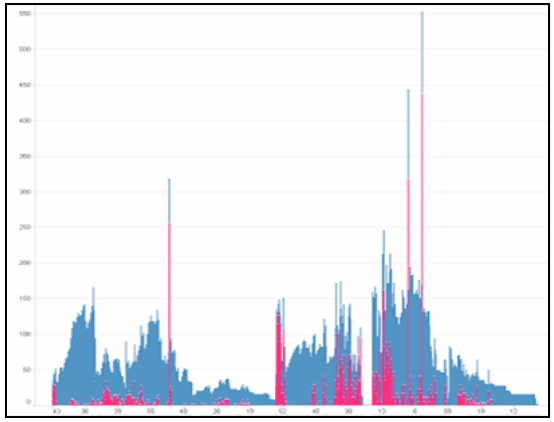


figure 1: Coding (blue) and chatting (pink) date stamps show simultaneous activities

Figure 1 illustrates the code date stamps marked in blue (compiled date stamps from the server logs) and the chat date stamps marked in pink. When we disaggregate the graph into individual users' chat times and code compilation times, the results suggest that participants were indeed chatting and coding simultaneously during both sessions.

We also analyzed the number of codes generated by looking at the date stamps collected every two minutes by the logger and self compiled programming statements. Figure 2 illustrates the lines of codes over time for one participant, Cassandra. Cassandra generated a lot of code on the first day but on the second day and third day, she decided to start over. From researcher's observation, she decided to use notepad to save portions of code that she thought she might want to use for later and utilized the cut and paste functions. We can reveal more interesting programming behaviors and themes from a preliminary manual content analysis of the type of messages sent.

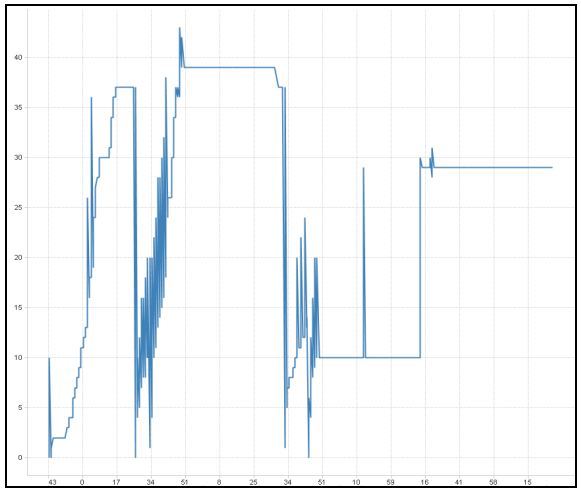


figure 2: Lines of code over time for one participant

Help-seeking and Social Support

We observed that participants sometimes asked one another for help as well as reported what they were doing currently (i.e. "I'm trying the CircleChat right now") as if to elicit shared situated support [1]. The following is an example of how two participants collaborated to successfully program the sound feature.

Jennifer: Type the animal that is commonly known to chase mice!
Cassandra: cat
Jennifer: awh….. my computer was supposed to make

[1] Names changed to protect identity

Serena is chatting and programming at the same time.

noise.
Jennifer: so sad.
Jennifer: Hmm…. Cat
Cassandra: maybe it only works when you type it
Jennifer: nope. Still no noise.
Jennifer: cat…. cat… nope….
Cassandra: check your code
Jennifer: OkeyDokey!
Jennifer: kitten… Oh well…. Can you see the smiley?
Cassandra: let me try…. probably
Cassandra: cat
Cassandra: see now it works…. try it now
Jennifer: cat

When we index this conversation into the date stamps in the client code, we can see that Jennifer had typed:

```
if teh chat text contains "cat" then play
"meow.wav"
```

The debugger did not catch the misspelling of the word "the" and Jennifer was stuck. At this instance, we see that Cassandra had already successfully typed in six different sound programming statements and could help Jennifer debug hers. Even though Jennifer and Cassandra were sitting next to each other, they used the chat client and verbally to test functions and to support one another to make the sound effect function correctly.

In some cases, we observed a tension between chatting and programming, where participants preferred not to do both at the same time:

Allison: heeeeeeeeeeeeeeeeeelllllllllllllllllllooooooooooo
Madison: can i talk 2 u wen i'm finished programing?
Madison: can i talk 2 u later
Allison: fine
Madison: i'm writing down codes

Again, they were in the same room, but used chat to reinforce a sense of presence and awareness of their programming activities. We also observed interplays between frontchannel and backchannel "collaboration in the air" [3]. After the exchange between Cassandra and Jennifer, the whole class heard them play the "meow" sound and asked them aloud "What was that?! How did you do it?" The girls would then flip through the code handout to look for a way to add sound to their chat or ask the researcher for help.

Programming and Play
Programming sometimes became playful. Brooke programmed a sound file "ugh.wav" to play when typing the word "ugh". Brooke would then type or say a boy's name she thought was cute and then Allison would respond by typing the word "ugh", initiating the "ugh" sound and both of them would laugh. This burst of engagement between them lasted for half an hour (and at the irritation of other participants in the room.)

Bruckman attributes part of Storm and Rachel's motivation to learn with their ability to connect their activities to popular culture. Bruckman argues that serious learning is not inconsistent with play and asserts the importance of designing "situated support" into learning contexts [1]. However, our study was conducted in a semi-structured environment with a specific task. It is not clear in what ways the environment might be extended to youth's natural, informal use of the Web.

Conclusion and Future Directions

Programming a chat client for the OLPC utilized two contexts to motivate the Girl Scouts: designing for the OLPC and the ability to chat concurrently while programming. We examined the effects of programming while chatting and found preliminary results that point to how chat worked not only as a motivational piece, but as a tool for the girls to support one another in learning, collaboration, and encouragement.

We also did not measure learning of computer science content and how chatting and programming might differ from traditional classroom pedagogy. We observed that participants reflected on what they were doing on one level and whether or not they enjoyed the activities. However, there was less evidence of how the activities may influence long-term interests in computing related courses or careers. For future directions, we aim to explore how the role of the OLPC context influences participants' motivation, what programming concepts the users actually learned, a comparison of collocated chat classroom environments compared to a completely virtual environment, how the text programming interface differs from graphical user interfaces, and examine how the activity influences participants' desire to pursue computing.

Acknowledgements

This work is supported by NSF BPC #0634629.

References

[1] Bruckman, A. "Situated Support for Learning: Storm's Weekend With Rachael." *The Journal of the Learning Sciences* 9.3 (2000): 329-372.

[2] Eastmond, Evelyn. "New Tools to Enable Children to Manipulate Images Through Computer Programming." 2006.

[3] Harel, I., and S. Papert. *Constructionism*. ABLEX Pub. Corp.

[4] Jozefowicz, D. M., B. L. Barber, and J. S. Eccles. "Adolescent work-related values and beliefs: Gender differences and relation to occupational aspirations." *Paper presented at Biennial Meeting of the Society for Research on Child Development* (1993).

[5] Kafai, Y. B. *Minds in Play: Computer Game Design as a Context for Children's Learning*. Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, 1995.

[6] Kelleher, C., R. Pausch, and S. Kiesler. "Storytelling alice motivates middle school girls to learn computer programming." *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press New York, NY, USA, 2007. 1455-1464.

[7] Ken Perlin, M. F., and A. Hollingshead. "The Rapunsel Project." *Virtual Storytelling: Using Virtual Reality Technologies for Storytelling; Third International Conference, ICVS 2005 Strasbourg, France, November 30-December 2, 2005 Proceedings*. Springer, 2005.

[8] Lenhart, A., and Pew Internet & American Life Project. *Social Networking Websites and Teens an Overview*. Pew/Internet, 2007.

[9] Pane, J. "A Programming System for Children that is Designed for Usability." *Proceedings of the First ESP Student Workshop*. 1997. 15-22.

[10] Roschelle, J. (1992). Learning by Collaborating: Convergent Conceptual Change. *The Journal of the Learning Sciences*, *2*(3), 235-276.

[11] Vesgo, Jay. "Interest in CS as a Major Drops Among Incoming Freshmen." *Computing Research News* 17 (3) (2005).