NetsBlox: a Visual Language and Web-based Environment for Teaching Distributed Programming

Brian Broll, Péter Völgyesi, János Sallai, Ákos Lédeczi
Vanderbilt University
akos.ledeczi@vanderbilt.edu

March 23, 2016

1 Introduction

Wing [34, 35, 36] and others [7] have described computational thinking (CT) as a general analytic approach to problem solving, designing systems, and understanding human behaviors. The integration of CT within the K12 curriculum has also been argued for by the ACM committee on K12 education [11], and has become a focus of several researchers on educational computing [28, 26, 18, 10].

There are many efforts around the world to introduce children to computer programming, such as code.org, Khan Academy [16], LEGO Mindstorms and the Raspberry Pi [25]. Earlier efforts in the 1980s involved the LOGO programming language [22]. Visual programming languages have come to play a prominent role in this movement and have been used to teach children programming [18, 15, 26] as well as using computational modeling to teach and learn science [28, 3, 27]. However, most of these efforts focus exclusively on the ***computer*** and neglect an equally important concept, the ***network***. This is of course completely understandable: you need to learn how to program a computer before you can create networked/distributed applications. Nevertheless, the majority of computer applications we and our children interact with daily rely on the network to provide their functionality. The web, texting, Twitter, Facebook and other social networks, multiplayer games, Pandora, Netflix, Amazon, Siri, Google Maps and YouTube are just a few of the most popular examples. Even embedded systems are becoming networked at a rapid pace with cars and home automation being the prime examples. Teaching distributed programming then constitutes both a necessity and a great opportunity. It is a necessity, because distributed computing is rapidly becoming part of basic computer literacy. And it is also an opportunity, because children already use the technology every day and their natural curiosity will provide excellent motivation for them to learn more about it.

We believe that it is not enough to introduce computer programming into the K12 curriculum, but it also necessary to teach distributed computing concepts to children. At the college level, the ACM IEEE Computer Science curriculum (2013) [33] advocates introducing the following topics to CS students: asynchronous and synchronous communication, reliable and unreliable protocols, and the need for concurrency in operating systems. We argue that with the help of a carefully designed visual representation, an intuitive user interface and a sophisticated cloud-based infrastructure, it will be possible to teach some of the key underlying concepts of distributed computation to high school students. To this end, we have developed a new learning environment called NetsBlox which extends the Snap! visual programming paradigm and environment [31]. NetsBlox introduces a few carefully selected abstractions that enables children to create distributed computing applications.

2 Background

The literature on educational computing is rife with observations of children's difficulties with learning basic constructs of programming. Programming languages tend to have only a few components which are combined in many different ways, and students find it challenging to understand the semantic results of different combinations, especially to achieve particular goals [29, 32, 23]. When students try to assemble language elements, they often get confused with syntax problems as they struggle to understand semantic ones [23]. Thus, alleviating syntax issues helps students focus on the semantic problems [13]. In fact, research comparing learning in a more and a less syntactically strict language, Java and Python, respectively, attribute the greater success of students in Python to be a result of reduced syntactic complexity [9]. Another documented programming challenge in the literature is students' lack of understanding of computational processes. Many students do not understand how interpretation of traditional computer languages works, e.g., how control flows and how variables get updated [4]. Some research claims that visual programming languages can make these processes more accessible by making flow of control constructs more natural.

Alleviating syntactic complexity is an important pedagogical affordance of a visual programming paradigm. In such an environment, students construct programs using graphical objects on a drag-and-drop interface [15]. This significantly reduces students' challenges in learning the language syntax (compared to text-based programming), and thus makes programming more accessible to novices. Examples of some visual programming environments are Snap! [12, 31], which itself is an extension of Scratch [18], AgentSheets [26], StarLogo TNG [17], ViMAP [27] and Alice [5]. We have built NetsBlox upon Snap! because it is one of the most mature and widely used approaches, it is open source and we have significant experience using and teaching it.

2.1 Understanding Concurrency

The visual formalism of Snap! and Scratch constitutes an excellent starting point because it supports concurrency [21, 19]. Sprites run in parallel and each script runs in its own thread. The keyboard and the mouse generate events that scripts can handle and scripts can generate and handle custom events. NetsBlox builds on these concepts to supply primitives for synchronization and communication ***across computers*** providing a gentle introduction to distributed computing.

Although researchers have been investigating approaches for teaching and learning computer programming in K12 classrooms, very few studies have looked at how K16 students can learn about concurrency. A few researchers have pointed out that Scratch can be used productively to introduce basic ideas of concurrency to novices [21, 19]. Meerbaum-Salant et al. [21] investigated difficulties experienced by students in understanding concurrency using Scratch. They divided this concept in two categories: Type I concurrency occurs when several sprites are executing scripts simultaneously, such as sprites representing two dancers. Type II concurrency occurs when a single sprite executes more than one script simultaneously; for example, a Pac-Man sprite moving through a maze while opening and closing its mouth. They found that Type I concurrency seems to be much more intuitive for students and easier to grasp.

Maloney, et al. [19] reported on an experiment where Scratch was used by students in an after-school clubhouse. These students were self-selected and self-paced, receiving no formal instruction. By analyzing the students projects, they found that the majority of the projects that actually constructed executable scripts used both sequential and concurrent execution. However, as the authors themselves note: without realizing it, most Scratch users make use of multiple threads (p. 368, our emphasis). The researchers did not investigate if the use of concurrency demonstrates understanding of this concept.

2.2 Motivation

We believe that the main appeal of NetsBlox is the increased motivation it provides because young learners will be able to create new classes of programs that are currently out of reach. For example,

multiplayer video games are very popular with children and NetsBlox supports the creation of non-trivial gaming programs. Real-time games with 3D scene rendering are obviously beyond the realm of possibilities. But strategy games, turn-based board games and games that include slower paced animation are quite feasible. In addition, NetsBlox applications can be hosted on phones and tablets. Just imagine an average high school student creating a multi-player game, running it on her phone and playing against a friend over the Internet after just a few weeks of instruction. That is the promise of NetsBlox.

Furthermore, there are a large number of publicly available interesting data sets on the web. Examples include the weather [37], air pollution [1], seismic data [6], astronomy [30], real-time traffic information [20] and many others. Typically the data is visualized on a given website, but in many cases a public API is available to access the data programmatically. The NetsBlox server already provides access to a select set of interesting data sources. These are available from NetsBlox programs via a simple abstraction called Remote Procedure Call (RPC). Essentially these services provide a mapping between NetsBlox RPC calls and the corresponding API of the public data service. For example, the NetsBlox Weather Service has a remote procedure called "*temperature at*" that takes arguments for the place and time and returns the corresponding temperature. In the background, it silently invokes the proper call on the OpenWeatherMap API to get the data.

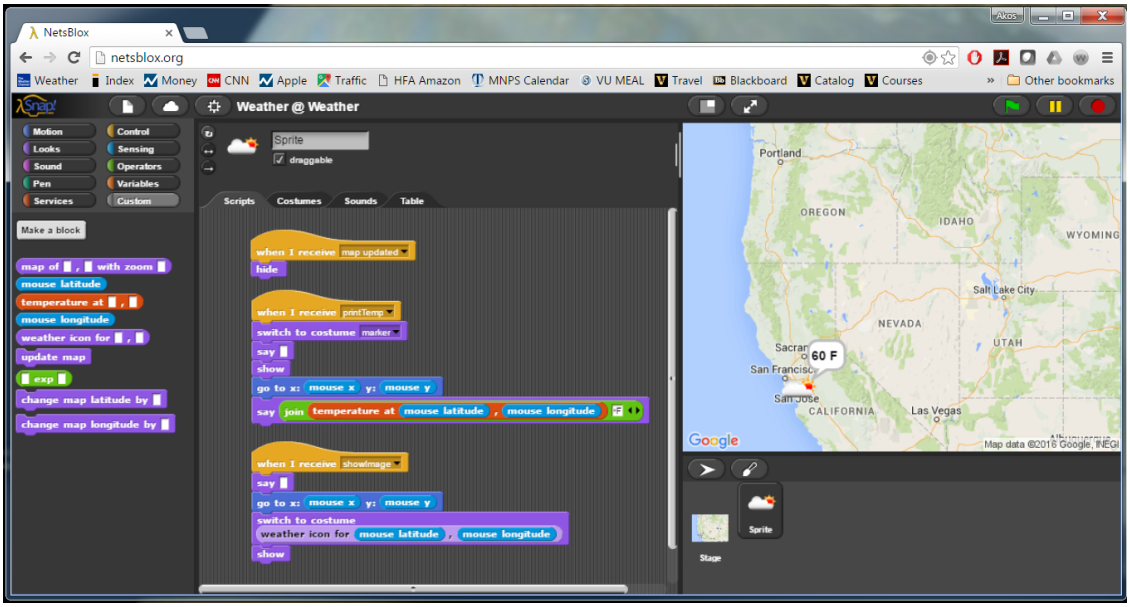

Figure 1: Weather Application in NetsBlox

The possibilities are quite literally limitless. With NetsBlox, children are able to create all kinds of imaginative applications that utilize the wealth of information provided to them using a single, simple abstraction. One potential difficulty is that much of the data are geospatial. To help children make use of it, NetsBlox integrates Google Maps as an interactive background. See Figure 1. Displaying real-time data on an interactive map using a Scratch-like easy-to-use visual programming language is one of the most attractive features of NetsBlox.

3 Distributed Programming Primitives

The key design decision for NetsBlox was the selection of distributed programming primitives manifesting themselves as visual abstractions. In order for the students to engage with the technology and be able to learn the basics of distributed computation, these needed to be intuitive, easy-to-grasp and show the essence of important concepts while hiding unnecessary complexity.

The two distributed programming primitives NetsBlox supports are *Messages* and *Remote procedure Calls (RPC)*.

Communication is supported by ***Messages***. Messages are very similar to Events already present in Snap! Basically, a Message is an Event that contains data payload. Users will be able to drag and drop one or more variables on the "`send msg`" block (called `broadcast` for events in Snap!). On the receiver side, when they pick the given message from the list of available ones, these data items will appear in the "`When I receive`" block header showing the appropriate names, as shown in Figure 2.

Figure 2: Sending and receiving messages with data in NetsBlox

Remote Procedure Call (RPC) is the highest level of distributed abstraction NetsBlox employs. Snap! supports the specification of Custom Blocks which are essentially functions. The NetsBlox server provides remote procedures (functions appearing as custom blocks) accessible via RPC. This is necessary for getting access to public scientific databases and it is also very helpful in facilitating multiplayer games. For example, the client code may invoke an RPC to make the next move or get the number of players currently in the game.

The semantics of RPC is as expected: multiple input arguments, single output argument, pass-by-value and blocking call. These should be familiar to users of custom blocks. In NetsBlox, we will provide an additional facility: a (script) local variable called "`error`" represented by a red block. If the RPC was successful, the value of `error` will be the string "OK." Other values will indicate various problems with the call, such as "timeout" or "bad argument". In a curriculum using NetsBlox, the recommended way of using RPCs need to be explained: after a call, an if statement should check whether the value of the error variable is OK or not. If not, the user program is expected to handle the error according to the value of the `error` variable.

The final distributed programming primitive is the concept of a ***Room***. A Room consists of ***Roles*** which are named NetsBlox clients. That is, a Room defines the NetsBlox clients which share a network and provides names for each client to facilitate messaging between clients. An example of this can be seen in Figure 3.

Figure 3: Sending messages to other NetsBlox clients

4 Illustrative Examples

To illustrate some of the concepts introduced above, let us consider a simple example. Figure 4 shows a trivial dice game. The idea is that two players both roll their dice and whoever has a higher number wins. In case of a tie, they roll again.

The program is symmetrical in that both players use the exact same scripts. Whichever player clicks the green flag first starts the game. The script corresponding to the green flag clicking event broadcasts an event to another script that rolls the dice by picking a random number between 1 and six and sends a message to the other player with the result. This very same script will be

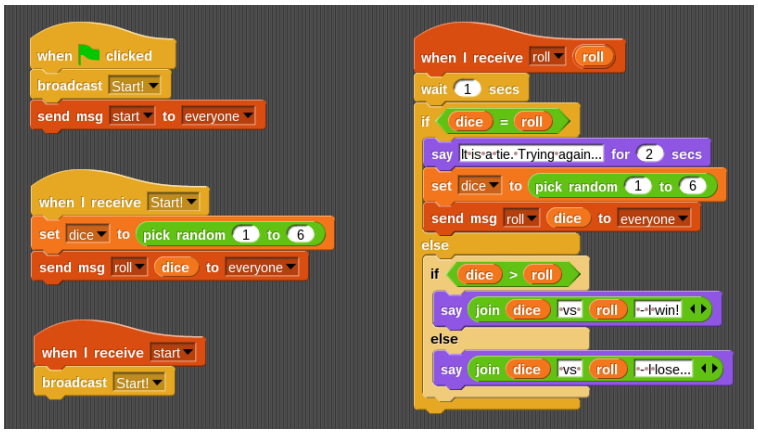

Figure 4: The scripts of the Dice game

executed by the other player too. This occurs because the green flag script also sends a message to the other player. When that message is received, its handler script broadcasts the same Start event which in turn, rolls the dice and send the value out to the first player.

When the message with the dice value arrives at either player, the corresponding script waits one second to perform lazy synchronization, i.e., to make sure that both players are in the same place. Then the script proceeds to compare its own dice value with the one that it received from the other side. If they are the same, it "rolls" again and sends the new value. Otherwise, the winner is declared with a text display on the stage.

It is interesting to note the message addressing in this example. All send blocks here used "everyone" as the addressee meaning every other role (player) in the current room (game). In a two-player game such as this, it simply means the single other player. Alternatively, one could select the name of the other role, but that would have resulted in slightly different scripts for the two players.

As a second example, consider an application that displays historical earthquake data. The program applies a combination of an RPC and messages. See Figure 5.

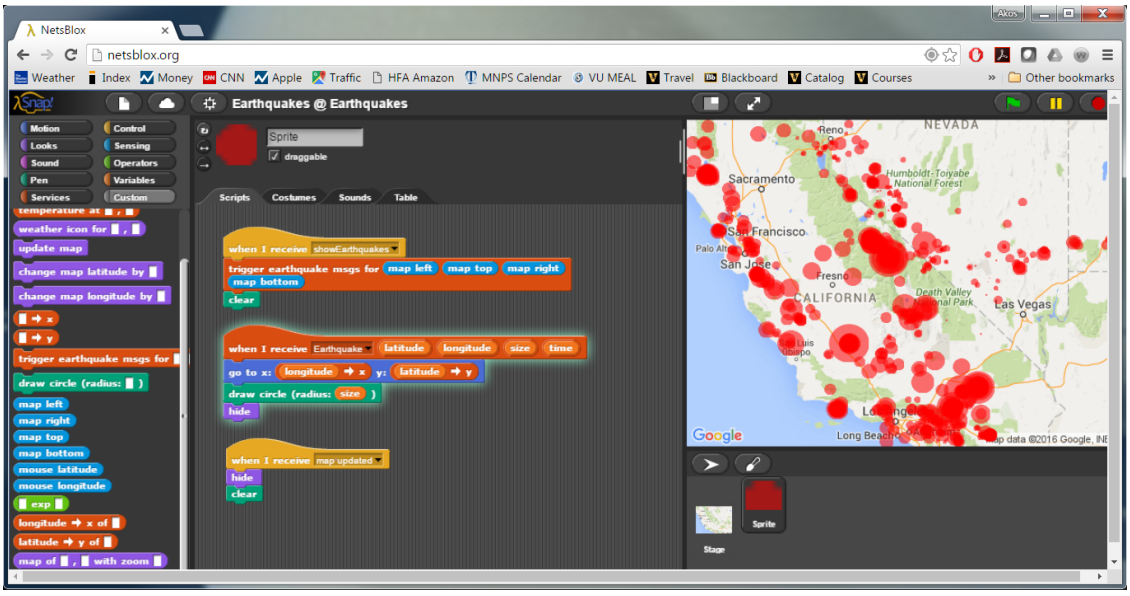

Figure 5: The Earthquake application

The application utilizes the interactive Google Map service just like the weather example in Figure 1. When the user clicks on the background, the program invokes the "trigger earthquake msgs" RPC. The server, in turn, collects the historical earthquake data from the web for the given geographic area and starts sending messages back to the client. One message per earthquake event is sent. The script that handles the Earthquake message simply displays the location with a red dot where the size of the dot is proportional to the magnitude of the given earthquake.

Finally, let us present a more complicated example: a two-player Tic-Tac-Toe game. We implemented it exclusively with peer-to-peer messages with no server support (in the form of RPCs). This means that the protocol required by the turn-based nature of the game needs to be implemented by the clients using messages only. We envision that for novice NetsBlox users, the environment will provide services to help with games, such as "two-player turn-based game" service or even specific helpers such as those that enforce the rules of a card game like poker. In fact, we have a much simpler implementation of Tic-Tac-Toe where the server decides whose turn it is and checks the status of the game, etc. The purpose of this exercise was to see how complicated the code for a simple game can get. In addition, this version uses a single sprite for one cell of the game board and cloning. It checks the game status on the client as well. Finally, it is symmetrical, that is, both clients run the exact same set of scripts. In other words, this is as complicated as a distributed Tic-Tac-Toe game can get.

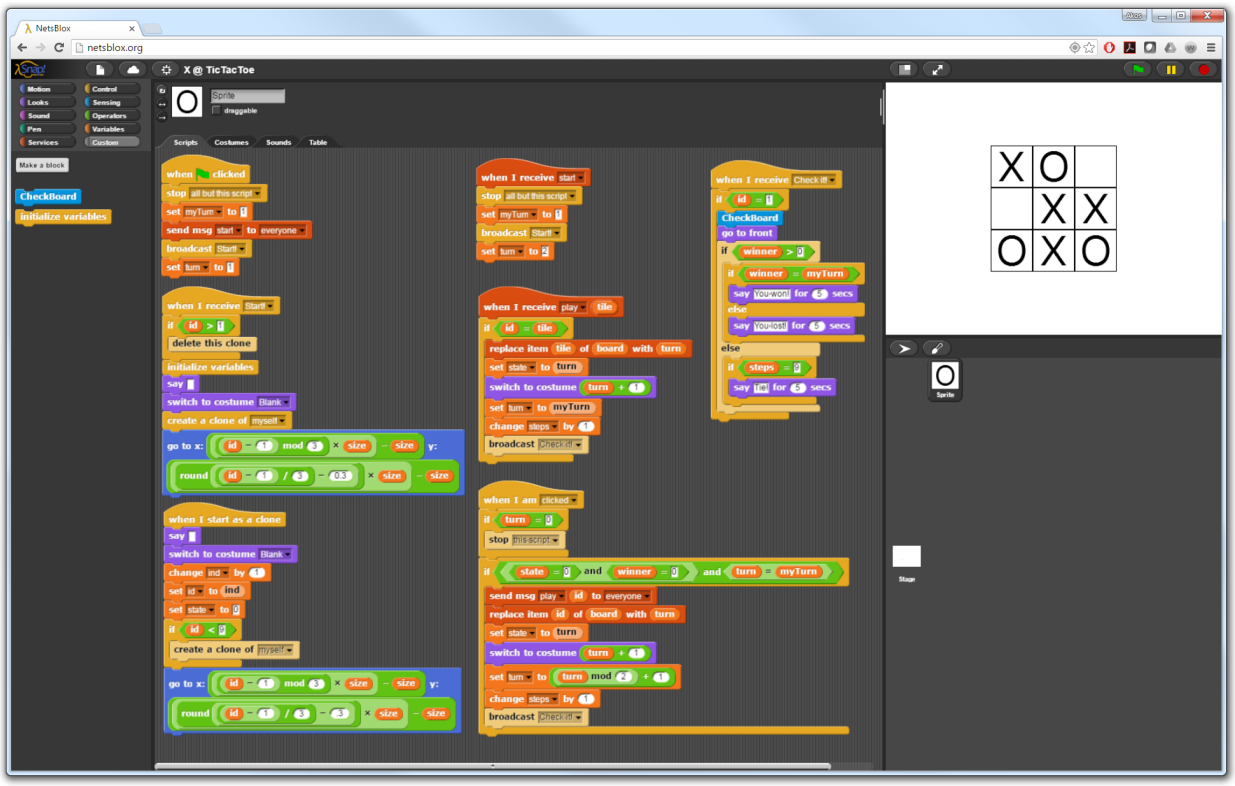

Figure 6: Tic-Tac-Toe Game

The end result is shown in Figure 6. Note that other than the "CheckBoard" and "initialize variables" custom blocks which are not complicated, all the scripts are shown in the figure. While a detailed explanation of the code is beyond the scope of this paper, note that only two message types are used: the *start* message that triggers the initialization of the game on the other client and the *play* message that communicates the latest move of one player to the other. Our conclusion after developing this game is that novice users will definitely need server side help to create multi-player

games. However, advanced students should be able to design, implement and debug programs of this complexity.

5 Network Model

NetsBlox requires a powerful yet easy-to-understand network model. It is important to note that the technical details described here are completely hidden from users. The core design principles of the network model are:
(1) reducing the accidental complexities of distributed programming by eliminating tedious and error prone tasks, shielding the programmers from distracting technical details and common pitfalls that include firewalls, routing, address resolution, automatic reconnection, etc., (2) uniform addressing and discovery scheme for distributed artifacts, and (3) support for both synchronous and asynchronous communication.

We opted for ***virtual overlay network abstraction*** to achieve these goals. This facilitates direct, bidirectional, peer-to-peer communication between NetsBlox software artifacts, as if they were part of a local area network, without the need for explicit message routing. This overlay network is built on top of existing network technologies, but without one-to-one mapping between virtualized primitives and actual network packets and connections. For example, the NetsBlox server infrastructure emulates peer-to-peer message exchange between two remote scripts using reliable connections between several nodes (i.e., two clients and the server) to provide robust communication, as well as to enable instrumentation and centralized management capabilities.

Containers: To extend the platforms supported beyond just web browsers, NetsBlox introduces the concept of a container. A container is a hosting environment for applications that provides a well-defined set of services, including network connectivity. NetsBlox provides containers for web-browsers, iOS and Android devices. In the future, the container concept will be extended to support networked embedded devices, such as the Raspberry PI, as well as server containers running in the cloud. In contrast to web-based or mobile clients that may come and go, cloud containers could be always on, making them an ideal choice to implement certain online services in the future.

Coordination: While it is be fairly straightforward to write a simple two-player game using the *Message* primitive (see Section 4), however, games with more complicated rules, such as chess, or with more than two players, such as bridge, can quickly become too complicated for the budding NetsBlox programmer to design and implement without help. In addition, these games require explicit server support, for example, to pair up users who do not know each other, but want to play the same game. If different applications (e.g. alternative game clients developed by the students) need to communicate with each other, they need to follow a common communication protocol. Protocol definition includes (1) protocol syntax, i.e., message formats and RPC signatures, and (2) protocol semantics textually describing how a service behaves.

6 NetsBlox Infrastructure

To support the network abstractions described above, distributed programming primitives (Section 3) and the overall application life-cycle, we have developed and deployed a cloud-based infrastructure and an easy-to-use web application.

The core infrastructure services include (1) hosting the web-based development environment, (2) data and application persistence, (3) RPC and message delivery services, (4) endpoint addressing, search and discovery and group communication and (5) application provisioning and instrumentation services. We host all elements of this infrastructure on the Amazon Web Services [2] cloud computing platform and provide all software components and deployment know-how on GitHub [8] with MIT open source licensing.

For supporting all communication primitives provided by NetsBlox, we propose to implement a

h

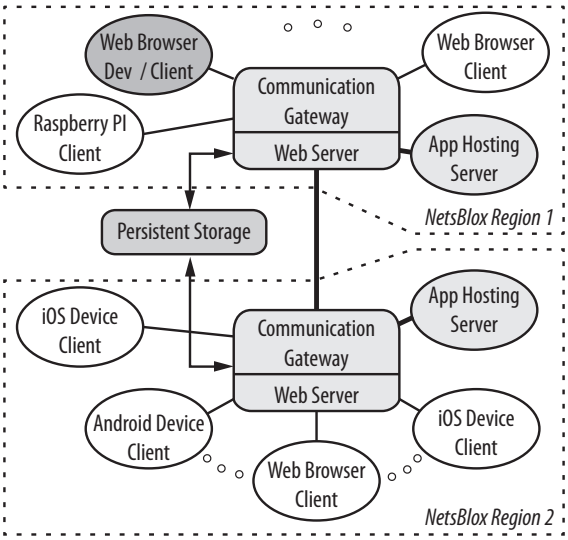

Figure 7: Network Architecture

virtual network abstraction—introduced in Section 5—with a decentralized client-server architecture, inspired by the XMPP/Jabber protocol [14]. In this model, each node initiates a connection to and registers itself with one of the server nodes. All further communication (administrative and management messages as well as application packets) are routed through this connection. While the primary deployment model targets a single classroom of students at a time, the proposed multi-homed architecture can scale to multiple large groups of users to collaborate from highly diverse geographical / administrative regions.

The high-level architecture of NetsBlox is shown in Figure 7. Servers—implemented with server-side JavaScript technologies—provide web services and will host web applications (e.g., the development environment), which can be accessed by browser-based clients. Each active client maintains a connection to a communication gateway for using the proposed (virtual) network services. Applications, persistent data, configuration information, traces and logs will be stored in a Persistent Storage service. In a multi-homed deployment model, servers will connect to each other and to the Persistent Storage.

Containers share a common HTML5 / JavaScript codebase to host applications, which are adapted to mobile platforms using currently available technologies such as PhoneGap [24]. The hosting environment can be executed on-demand or registered as a background service to allow remote application deployment and provisioning.

7 Conclusions

The paper introduced a visual programming language and corresponding web-based development environment called NetsBlox. NetsBlox is an extension of Snap! and it builds upon its visual formalism as well as its open source code base. NetsBlox adds distributed programming capabilities to Snap! In particular, it introduces two simple abstractions: messages and Remote Procedure Calls (RPC). Messages containing data can be exchanged by two or more NetsBlox programs running on different computers connected to the Internet. RPCs are called on a client program and are executed on the NetsBlox server. These two abstractions make it possible to create distributed programs, for example multi-player games or client-server applications. The paper illustrated the power of these abstractions by demonstrating a weather application with an integrated Google Map

based interactive background and a simple two-player game.

Bibliography

[1] AIRNow: National air quality information website. `http://airnow.gov/`. Cited 2016 March 16.

[2] Amazon web services, cloud computing infrastructures. `http://aws.amazon.com/`, 2016.

[3] Blikstein, P., and Wilensky, U. An atom is known by the company it keeps: A constructionist learning environment for materials science using agent-based modeling. *International Journal of Computers for Mathematical Learning 14*, 2 (2009), 81–119.

[4] Boulay, B. D., O'Shea, T., and Monk, J. The black box inside the glass box: presenting computing concepts to novices. *Int. J. Hum.-Comput. Stud. 51*, 2 (Aug. 1999), 265–277.

[5] Conway, M. J. Alice: Easy–to–Learn 3D Scripting for Novices. Master's thesis, University of Virginia, Faculty of the School of Engineering and Applied Science, December 1997.

[6] EarthScope: Complementary seismic data sets. `http://www.earthscope.org/science/data/complementary/`. Cited 2013 November 1.

[7] for the Workshops on Computational Thinking; National Research Council, C. *Report of a Workshop on The Scope and Nature of Computational Thinking*. The National Academies Press, 2010.

[8] GitHub - Web-based Project Hosting. `http://github.com/about`. Cited 2013 November 1.

[9] Grandell, L., Peltomäki, M., Back, R.-J., and Salakoski, T. Why complicate things?: introducing programming in high school using python. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52* (Darlinghurst, Australia, Australia, 2006), ACE '06, Australian Computer Society, Inc., pp. 71–80.

[10] Guzdial, M. Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments 4*, 1 (1994), 001–044.

[11] Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., and Hosking, A. L. A multidisciplinary approach towards computational thinking for science majors. In *Proceedings of the 40th ACM technical symposium on Computer science education* (New York, NY, USA, 2009), SIGCSE '09, ACM, pp. 183–187.

[12] Harvey, B., and Mönig, J. Bringing no ceiling to scratch: can one language serve kids and computer scientists. *in Proc. of Constructionism* (2010), 1–10.

[13] Hohmann, L., Guzdial, M., and Soloway, E. Soda: A computer-aided design environment for the doing and learning of software design. In *Computer Assisted Learning*, I. Tomek, Ed., vol. 602 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1992, pp. 307–319.

[14] Hornsby, A., and Walsh, R. From instant messaging to cloud computing, an XMPP review. In *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on* (2010), pp. 1–6.

[15] Kelleher, C., and Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv. 37*, 2 (June 2005), 83–137.

[16] Khan Academy: Non-profit educational website . `https://www.khanacademy.org/cs`. Cited 2016 March 16.

[17] Klopfer, E., Yoon, S., and Um, T. Teaching complex dynamic systems to young students with starlogo. *Journal of Computers in Mathematics and Science Teaching 24*, 2 (April 2005), 157–178.

[18] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. Scratch: A sneak preview. In *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing* (Washington, DC, USA, 2004), C5 '04, IEEE Computer Society, pp. 104–109.

[19] Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. Programming by choice: urban youth learning programming with scratch. In *ACM SIGCSE Bulletin* (2008), vol. 40, ACM, pp. 367–371.

[20] MapQuest Traffic API. `http://developer.mapquest.com/web/products/dev-services/traffic-ws`. Cited 2016 March 16.

[21] Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M. Learning computer science concepts with scratch. *Computer Science Education 23*, 3 (2013), 239–264.

[22] Papert, S. *Mindstorms: children, computers, and powerful ideas.* Basic Books, Inc., New York, NY, USA, 1980.

[23] Perkins, D. N., and Simmons, R. Patterns of misunderstanding: An integrative model for science, math, and programming. *Review of Educational Research 58*, 3 (1988), pp. 303–326.

[24] PhoneGap: Open Source Cross-platform Mobile Application Framework. `http://phonegap.com/about/`. Cited 2013 November 1.

[25] Raspberry Pi: Low-cost single-board computer. `http://www.raspberrypi.org/`. Cited 2013 November 1.

[26] Repenning, A. Agentsheets: a tool for building domain-oriented visual programming environments. In *INTERCHI* (1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. N. White, Eds., ACM, pp. 142–143.

[27] Sengupta, P., Farris, A. V., and Wright, M. From agents to continuous change via aesthetics: Learning mechanics with visual agent-based computational modeling. *Technology, Knowledge and Learning 17*, 1-2 (2012), 23–42.

[28] Sengupta, P., Kinnebrew, J., Basu, S., Biswas, G., and Clark, D. Integrating computational thinking with k-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies 18*, 2 (2013), 351–380.

[29] Shneiderman, B., Mayer, R., McKay, D., and Heller, P. Experimental investigations of the utility of detailed flowcharts in programming. *Commun. ACM 20*, 6 (June 1977), 373–381.

[30] Sloan Digital SkyServer Website. http://cas.sdss.org/dr6/en/proj/. Cited 2016 Mar 16.

[31] Snap!: a visual, drag-and-drop programming language. http://snap.berkeley.edu/snapsource/snap.html. Cited 2016 March 16.

[32] Spohrer, J. C., and Soloway, E. Novice mistakes: are the folk wisdoms correct? *Commun. ACM 29*, 7 (July 1986), 624–632.

[33] The Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), I. C. S. Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. http://www.acm.org/education/CS2013-final-report.pdf, 2013.

[34] Wing, J. M. Computational thinking. *Communications of the ACM, Viewpoint 49*, 3 (Mar. 2006), 33–35.

[35] Wing, J. M. Five deep questions in computing. *Communications of the ACM, Essay 51*, 1 (Jan. 2008), 58–60.

[36] Wing, J. M. Computational thinking: What and why? *Link Magazine - Carneige Mellon University School of Computer Science* (Nov. 2010).

[37] Weather Underground: meteorological datasource. http://www.wunderground.com/weather/api/. Cited 2016 March 16.