Removing the Walls Around Visual Educational Programming Environments

Brian Broll*, Ákos Lédeczi*, Gordon Stein*, Devin Jean*, Corey Brady*
Shuchi Grover†, Veronica Catete‡, and Tiffany Barnes‡
*Vanderbilt University, Nashville, Tennessee, Email: akos.ledeczi@vanderbilt.edu
†Looking Glass Ventures, Palo Alto, California, Email: shuchig@cs.stanford.edu
‡North Carolina State University, Raleigh, North Carolina, Email: tmbarnes@ncsu.edu

***Abstract*—Many block-based programming environments have proven to be effective at engaging novices in learning programming. However, most restrict access to the outside world, limiting learners to commands and computing resources built in to the environment. Some allow learners to drag and drop files, connect to sensors and robots locally or issue HTTP requests. But in a world where most of the applications in our daily lives are *distributed* (i.e., their functionality depends on communicating with other programs or accessing resources and data on the internet), the lack of support for beginners to envision and create such distributed programs is a lost opportunity. This paper argues that it is not only feasible, but crucial, to create environments with simple yet powerful abstractions that open up distributed computing and other widely used but advanced computing concepts including networking, the Internet of Things, and cybersecurity to novices. By thus removing the walls around our environments, we can expand opportunities for learning considerably: programs can access a wealth of online data and web services, and communicate with other projects. Moreover, these changes can enable young learners to collaborate with each other during program construction whether they share their physical location or study remotely. Importantly, providing access to the wider world will also help counter widespread student perceptions that block-based environments are mere toys, and show that they are capable of creating compelling applications. The paper presents NetsBlox, a programming environment that supports these ideas and shows that tools can be designed to democratize access to powerful ideas in computing.**

I. Introduction

There are many block-based educational programming environments designed to make programming accessible to novices. With inspiration from Logo, many have been very effective at teaching programming to young learners. However, many of them keep students confined within the tool. This paper argues that removing these walls would be highly beneficial, both, to be able to teach more advanced concepts and to broaden participation in computing among young learners.

We believe that it is desirable to provide uniform support, in the form of a few intuitive abstractions, to open up block-based programming environments in such a way that students can create truly distributed applications. First, modern environments need to be easily extensible, and afford loosely coupled, easily-discoverable methods of integration with external resources such as web APIs. Adding a new resource should require no code changes or user interface changes on the client (i.e., no new blocks). This not only reduces the implementation effort required but also presents the external resources in a uniform, predictable way to the young learners.

Second, environments should support methods of communication between projects. Distributed computing is ubiquitous both generally and in applications popular among today's youth. Block-based environments, designed to make computing accessible and engaging, seem to be missing a crucial opportunity when they restrict learners from creating "social" applications that leverage the internet for communications.

Finally, collaborating with peers can be fun and engaging and can also improve learning [1]. Furthermore, collaboration and teamwork are vital parts of industry applications. Supporting equitable collaboration that goes beyond co-located pair-programming helps promote engagement and valuable 21^{st} century skills, and also dispels misconceptions about software being developed in isolation.

For the rest of the paper, we use the open source NetsBlox tool [2], [3] to demonstrate how advanced computing concepts can be made accessible to novice programmers. At the same time, some of these very same features make it possible for students to both develop the programs in a collaborative fashion and debug, test and use them together whether they are in the same classroom or studying from home.

II. Related Work

Scratch [4] is arguably the most popular tool among block-based programming environments. Designed for younger learners, Scratch led the field in making programming accessible through visual programming, immediate effects, and affordances that help prevent programming errors. It facilitates the creation of "Scratch extensions" with blocks that bring new capabilities to the environment, including language translation and support for interacting with a number of physical devices, such as Micro:bit and Makey Makey. At the time of this writing, there are 11 supported extensions: 6 for interacting with physical devices, 2 related to language, and 3 providing custom blocks for local capabilities such as drawing or playing music. With each of these extensions, Scratch brings in a number of new blocks, which can make it harder to find blocks and may steepen the learning curve. Scratch supports

978-1-6654-4592-4/21/$31.00 ©2021 IEEE

limited distributed data sharing via Cloud Variables that enable instances of the same program to share variables.

Snap*!* is a conceptual descendant of Scratch designed to support more advanced features including first class lists and functions, as well as richer support for custom blocks [5]. Snap*!* also allows for extensions, e.g., to physical devices via libraries, and provides a block for making HTTP requests. However, processing the information returned by such requests is anything but intuitive, thus adding unnecessary complexity that block-based environments are designed to remove in the first place.

BlockyTalky [6] supports the development of distributed applications for devices like the Raspberry Pi and Micro:bit. It facilitates communication between the devices allowing network messages which can be sent to a given IP address and port, but it does not support generic internet access or the creative programming elements of Scratch & Snap*!*.

MIT App Inventor is designed for development of mobile applications [7] and consists of "Designer" and "Blocks" editors. The "Designer" editor is used to add components to the app and the "Blocks" editor is used to program the app's behavior. App Inventor has native support for HTTP requests, Lego Mindstorms, and Firebase [8]. Additional capabilities are supported using "extensions" that consist of new types of components and their corresponding blocks similar to Scratch. In contrast, NetsBlox uses a single self-documenting block (called "call") to provide access to a large number of online services and WiFi enabled hardware devices.

CloudDB, Internet-of-Things, and machine learning capabilities are supported as App Inventor extensions and enable apps to store data in the cloud, communicate with various devices like Arduino, and incorporate various ML-based pattern recognition capabilities. After adding a component from one of these extensions to an app, the user has the ability to configure the component accordingly. This may include providing a secret access token or URL for a web-based service. After the component has been added to the app, the corresponding blocks will be available in the palette of the Blocks editor. App Inventor also has some support for real-time collaboration and merging projects.

A recent addition to the App Inventor toolbox is support for the creation of Alexa skills; although currently this is through a forked version of the environment. This version changes the editor to add new programmable entities (i.e., Alexa skills) and provides a chat dialog for testing it. Creating Alexa skills in App Inventor is exciting, but achieving it by modifying the editor itself is not scalable. NetsBlox required neither a user interface change nor any new blocks to add a similar capability.

Today, web services are becoming a required topic to teach in some high school computer science curricula [9]. Lim et al. believe that web services should begin to be taught in introductory computer science classes [10]. However, there have been difficulties in teaching web services without proper tools [11]. Instructors Assunção and Osório found that when teaching web services to computer science undergraduates, students focused more on issues involving the configurations of tools for the course instead of the actual material [11]. With its simple implementation of the Remote Procedure Call (RPC) "call" block, NetsBlox allows web services to be taught as an easy-to-comprehend concept. The "call" block eliminates tooling issues and allows novice programmer students to focus more on the subject matter at hand without being overwhelmed—a key notion in introductory computer science programming classrooms.

While tool support for collaboration is generally lacking in educational programming environments, educators still try to encourage their students to work together. For example, collaboration in Snap*!* for the popular Beauty and Joy of Computing course is encouraged through side-by-side driver-navigator pair programming [12]. This paradigm requires the driver to make edits to a program, while the navigator monitors the progress (e.g., by reading instructions or requirements). Built-in tool support in NetsBlox enables such pair programming without having to be co-located, and it also opens the door for other models of collaboration.

In summary, most existing environments lack 1) a uniform and intuitive way to access resources on the internet, 2) general support for distributed applications and 3) flexible, synchronous and asynchronous collaboration support.

III. Online Data and Web Services

NetsBlox introduces a simple abstraction to provide access to a set of selected online data sources and web services. *Remote Procedure Calls (RPC)* allow users to invoke functions running remotely on the NetsBlox server and provide results as return values. Related RPCs are grouped into *Services*. Examples are Google Maps, Weather, Earthquakes, the Movie Database, and many others. Additional services that run directly on the NetsBlox server, and do not require third party support, include a Gnuplot-based chart service and a hierarchical key-value store called Cloud Variables.

How much complexity does it involve to have this much functionality available? Won't users get overwhelmed and confused by this? RPCs use a ***single block*** called "call." Furthermore, the block is self documenting. It has two pull-down menus, one for the service and one for the RPC. When a service is selected, the second menu reconfigures itself to show the RPCs available within the selected service. When an RPC is selected, slots for the required input arguments appear along with their names. See examples in Figure 1. In addition, service- and RPC-specific documentation is available by context clicking on the call block and selecting help.

Fig. 1: Example RPC calls

To illustrate the simplicity and intuitive nature of this abstraction, consider a 13-block program that shows a map

of the local area of the user and displays an icon representing current weather conditions anywhere the user clicks (Figure 2). It is not necessary to know anything about NetsBlox or read comments to understand what the code does and how it works. To make the background a pan-able and zoom-able fully interactive map of the world requires only 20 additional blocks.

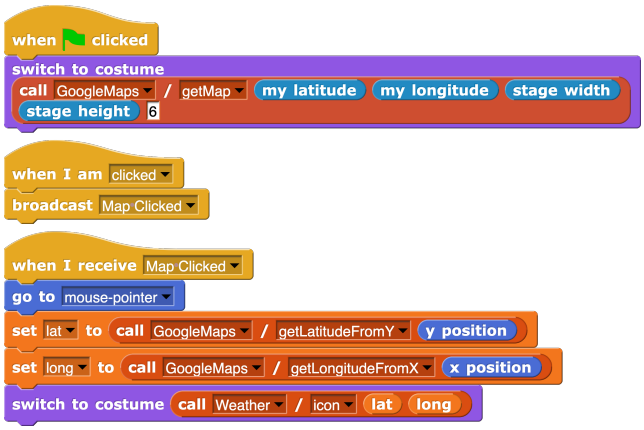

Fig. 2: Current weather conditions project. Top two scripts: stage. Bottom script: sprite.

The "call" block is a truly powerful abstraction. Using a single generic block that configures itself according to context removes the cognitive load of learning a new set of blocks for every service. It also eliminates palettes full of new and unfamiliar blocks that would require searching for just the right one. Another factor that makes the RPC concept familiar to students is that it closely resembles custom blocks. Both of them have multiple inputs and a single output, and are blocking calls that cause the program to wait for the result. The only differences are that RPCs run on the server and they are not user-defined.

RPCs return data in the form of numbers (e.g., temperature), text (e.g., city name), lists (e.g., movie IDs), multi-dimensional arrays (e.g., geolocation search results), or images (charts, maps, movie posters, etc.). These are built-in data types and students are already familiar with them. Users do not need to de-serialize the data, parse text, or a process a JSON data structure to extract useful information from results, unlike with HTTP calls available in other tools.

Services allow students to create projects that utilize a wide array of information freely available on the internet. Many of these are sources that students already use in their daily lives (e.g. maps, weather, movie ratings, etc.). Others are related to topics they may care about, such as climate change or sports. Helping students create projects tied to their interests and related to real world issues will increase their motivation to learn to program and make programming and computer science more relevant to them [13].

IV. Communication

Teenagers spend a lot of time on social media and with online multi-player games. What kind of support would a programming environment need to let them ***create*** such applications as opposed to just consuming them? Message passing is probably the most important abstraction in distributed computing. We picked it to enable NetsBlox projects running anywhere on the internet to be able to communicate with each other.

Messages in NetsBlox are very similar to events in Scratch and Snap!. Messages are more powerful, though, as they can carry data and they do not have to stay within the project (i.e., within the walls of the environment); they can travel to any other NetsBlox project running at the time of sending. Messages have types, defined by a name and the data the message is to carry (i.e., an ordered set of input slot names). Message type definition is done similarly to how one defines a custom block header in Snap!.

Only two blocks are needed for message passing: one for sending and one for receiving. Selecting a message type in the "send" block pull down menu reconfigures it to show the corresponding input slots with their names provided. Similarly, selecting a message type in the "when I receive" receiver hat block shows the same fields as variables, just like a custom block definition does, as shown in Figure 3.

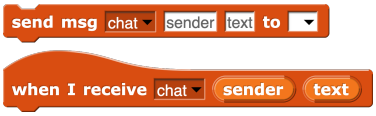

Fig. 3: Message passing blocks

Message data can be of any of the data types supported by the environment, even scripts that the receiver can later run! The data is not strongly typed, so the sender and receiver must agree on what the message means. If the two projects need a prescribed interaction pattern then they also need to agree on a sequence of different messages of typically different types. In other words, they have to design a protocol. But for simple applications, a single message type suffices. Consider Figure 4 showing a 6-block chat application (using the 'chat' message type defined in Figure 3), which allows two or more students to chat with each other, each running the same project shown.

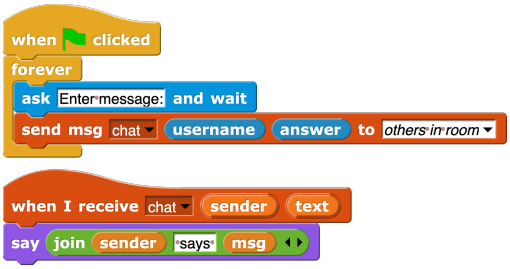

Fig. 4: Simple chat app

Another important concept in message passing is addressing. The sender must specify where to send the message. NetsBlox supports local addressing. A NetsBlox project can consist of multiple subprojects, each of which plays a *Role* in the project, while the project itself is referred to as the

Room. This naming convention comes from the fact that we anticipate many students will use message passing for creating multi-player games. Subprojects can be assigned to different users to run on different machines, e.g., play a game against each other. In turn, messages can be sent to any role or a group of roles within the same project. For example, in the chat application in Figure 4, the chat messages are sent by any one role to all the other roles, i.e., to *others in room*.

Messages can also be sent to any running application. A globally unique address is constructed by the (username, project name, role name) tuple, since each of these is guaranteed to be unique within its context. Global addressing is useful when one wants to support a dynamic number of participants in a distributed application. For example, the simple chat application could be extended to be an actual chatroom that users can dynamically join and leave. Another example would be peer to peer networking or a citizen science project, where a server (also a NetsBlox project) divides up a parallel complex problem into small tasks and distributes them to volunteer workers (running the same NetsBlox "client" project). The famous NASA SETI project [14], [15] is a real world example of this kind of distributed computing.

These examples illustrate that the message passing abstraction in NetsBlox has a low threshold, enabling students to write non-trivial applications with just a few blocks. But it also has a high ceiling, allowing students to create complex distributed applications in a block-based environment. The abstraction hides a lot of the *accidental* complexity associated with message passing and networking, but it exposes the most important concepts for distributed computing, including: message types, protocols, latency, and addressing.

Note that services can include messages and not just RPCs. For example, a call to the Earthquake service may need to provide data on thousands of earthquakes in the requested geographic area. Instead of returning a huge multi-dimensional array, the service sends one message per earthquake containing the date, magnitude, and location in separate fields. The N-Player game service that provides generic support for turn-based games also utilizes messages. For example, upon receiving an "end of turn" RPC call from one player, the service sends a message to the player whose turn is next.

V. Robotics Reimagined

The traditional approach to educational robot programming requires a local connection to the device via USB or Bluetooth to download the program to be executed on the robot. NetsBlox takes a different approach. WiFi-enabled robots can connect to the NetsBlox server directly via the internet. In turn, a service called RoboScape allows NetsBlox programs to send wireless commands to the robots. The robot runs a command interpreter that executes these remote commands. Robots respond by sending messages back to the user's program containing, for example, sensor values. This approach has several advantages. The student's code runs in the browser, making it much easier to test and debug. Student programs, and consequently, the robots, can communicate with each other and hence, collaborative robotics becomes feasible. Furthermore, the environment makes it possible to "overhear" other students' communication with their robots and hijack them. This motivates the need for cybersecurity and makes the subject much more tangible and fun to learn. We have carried out multiple successful and popular high school cybercamps built around wireless robotics [16].

Since the student's running program and the robot do not have to be co-located, remote robotics becomes possible. All one needs is a webcam streaming a video of the robot "arena" and multiple students can use the robots from their own homes. Remote robot programming can also be accomplished with more traditional Bluetooth-enabled robots that do need a local computer to interface with. In the Spring of 2020, after the pandemic started, Birdbrain Technologies set up multiple Hummingbird robots, connected them to a laptop where they run a NetsBlox program that controls the local robots using messages they receive from a remote NetsBlox project. Students are provided with various remote template projects that have the required message types already predefined and encapsulated in custom blocks. They can then use these blocks in their programs to control the robots remotely. The first command sends the "reserve" message that assigns the robot to the given user for a fixed amount of time if it is available. The robot action is live-streamed in another window [17].

Real physical robots are fun, and students love hands-on activities. However, robots are either inexpensive but simple, or powerful but expensive, and they can be hard to maintain, especially in a school setting. Furthermore, connecting WiFi devices to a school network can be problematic. Our work-in-progress project has created a flexible, modular, and immersive virtual robotics environment in which a group of students share an instance of a virtual world and each has their own virtual robots to program. The virtual robots are programmed just like the physical ones through the RoboScape service. Virtual robotics is an ideal setting to introduce advanced computing, because many of the constraints of physical devices can be controlled—either *removed* or, when pedagogically useful, *amplified* [18]. Yet, they are still tangible for today's youth accustomed to virtual experiences. Virtual robots can vary widely in their capabilities regarding locomotion, sensors, and actuators. They can contain the most advanced hardware from GPS to Lidar since there is virtually no cost associated with them. Virtual worlds can range from urban environments to deserts; from the open ocean to space. Students can view the entire virtual world on the classroom projector or their own computers/devices, or they can wear a VR headset to get a first-person view and a truly immersive experience. The environment is designed to be remotely accessible, so students can share worlds regardless of their geographic location. The environment looks just like a VR game, except that students create and program it, and not just play with it. We will conduct multiple summer camps for high school students where they will work on robot and cybersecurity challenges.

VI. Your Phone as a Sensor

Many schools offer makerspaces and other opportunities for students to get their hands on simple embedded computers, sensors and educational robots. However, most do not. Also, the kind of sensors and devices available are limited by cost. Finally, these kinds of activities are necessarily in-person, restricted to the school where the lab is located. However, almost every student in developed and even developing countries has a smartphone (or other mobile device) that contains a rich collection of sensors that are connected to the internet out of the box. This presents an opportunity to teach concepts related to the Internet of Things (IoT), networking, and distributed computing in a manner that is not only accessible to novices but also highly engaging and motivating. To make this approach a reality, we have created PhoneIoT, a mobile app which allows the built-in sensors of the device to be accessed remotely from NetsBlox. Since these devices have touchscreens as well, PhoneIoT makes it possible to configure a graphical user interface on the phone from the very same NetsBlox program that processes the sensor data and handles events from the mobile device. Hence, students can build truly distributed applications that run on two or more computers connected via the internet and that interact with the physical world via sensors.

Exercise Tracker

To illustrate how PhoneIoT can be used to create powerful and engaging projects, we present a simple exercise tracker which plots the user's route on top of Google Maps displayed in the NetsBlox client, streams the updated display back to the mobile device and prints the total distance covered as well. To emphasize the use of PhoneIoT's custom GUI controls, we also add start/stop buttons.

Figure 5 shows a portion of the initialization code. The device ID and password displayed in the PhoneIoT menu are manually entered in the NetsBlox client code for security. If the correct credentials are provided, PhoneIoT will accept configuration RPC calls such as the ones in the figure, which add controls at certain coordinates and dimensions. Adding the other controls is done with similar RPCs (not shown). The last block in the figure requests location data every 2 seconds.

Fig. 5: NetsBlox code to initialize communication with the PhoneIoT app and add GUI widgets on a mobile device

After initialization, we receive location updates via the "location" message type and begin plotting the course. Each message gives us the latitude and longitude (among other things), which we can convert into screen coordinates with the GoogleMaps coordinate translation RPCs. See Figure 6. The "getDistance" RPC of the GoogleMaps service can give us the distance between two locations, though we need to perform some averaging to reduce errors due to GPS inaccuracy [19]. This is done inside the "add point" custom block (i.e., function). The only other required logic is handling the stop and start events from the custom buttons on the phone. See Figure 7 for the final app screens in NetsBlox and PhoneIoT.

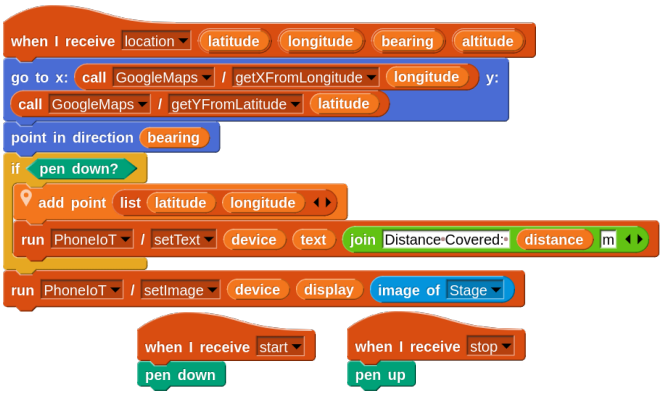

Fig. 6: Exercise tracker code. The "add point" custom block maintains the distance covered.

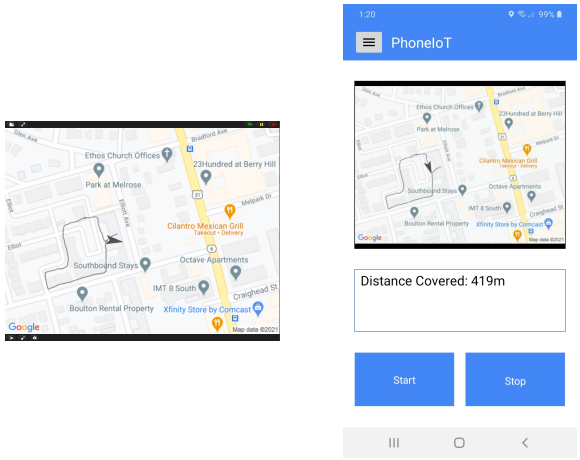

Fig. 7: NetsBlox client stage (left) and phone display (right) of the exercise tracker app at slightly different times

VII. Collaboration

Once we remove the walls around the programming environment, it becomes possible to support collaborative programming on projects. NetsBlox allows users to issue and accept invitations to collaborate on a project. Collaborators can then work on the same project simultaneously. Concurrent editing operations show up on everyone's screen. The server resolves conflicting changes by approving the first one, and rejecting subsequent ones. However, since the typical latency is under 100 milliseconds, this rarely happens.

Since NetsBlox stores only a single shared copy of the project, students can also work asynchronously. This is similar

to how popular collaborative editing tools such as Google Docs or Overleaf work. However, there is a conceptual difference between static documents and continuously executing block-based code. The latter has a state: variable values and the appearance of the sprites and the stage. Since each user's computer executes the code independently, the program state would be hard, if not impossible, to synchronize. So NetsBlox only keeps the program itself in sync across collaborators. The scripts will be the same, but the stage and the variable values will typically be different across users.

Collaboration support enables pair programming, team projects, remote tutoring, and remote collaboration. The latter two have been especially important with online learning in the middle of a pandemic. Furthermore, it also makes it possible to try out novel ways of teaching. For example, designing for collaboration can involve assigning subtasks to collaborating students. This can also help highlight problem decomposition—a key aspect of learning programming and computational thinking [20].

Yet another novel way to collaborate and share one's work with classmates and/or the teacher in NetsBlox is through Activity Galleries. Galleries enable members of a class or group to publish in-process and final-form NetsBlox projects to a shared space. Published entries preserve the projects' current state and also allow them to be viewed, commented on, loaded, and remixed by other group members. Finally, an Activity Gallery can be scoped to a particular classroom, so the group can benefit from seeing and giving feedback on each other's ideas as they emerge.

VIII. Extensibility

The NetsBlox environment has two major components: the client and the server. The client has unlimited undo and redo support and the capability to replay the entire history of the project. This also serves as simple version control, since one can go back to any past point in the history and continue from there. The NetsBlox client also adds a new block category to Snap*!* called Network, containing the RPC block and blocks related to message passing.

Unlike other environments, most NetsBlox functionalities are provided by the server, which runs the various services, routes messages, and manages collaboration (Figure 8). The architecture of the server is modular, facilitating extensibility. To add a new service, only a single JavaScript file (based on a template) needs to be added. Some are as simple as a few lines of code, while others providing more complex functionality can get large. But they are well separated, with a simple API connecting them to the core.

The power of this approach is illustrated by the fact that adding support for hardware devices in the form of the RoboScape service that manages WiFi connected robots or the PhoneIoT app that connects to mobile devices required no change on the client side or the server core at all. Most importantly, students do not have to learn any new blocks when a new service is added. All they see is a new, self-documenting option in the pull down menu of the "call" block.

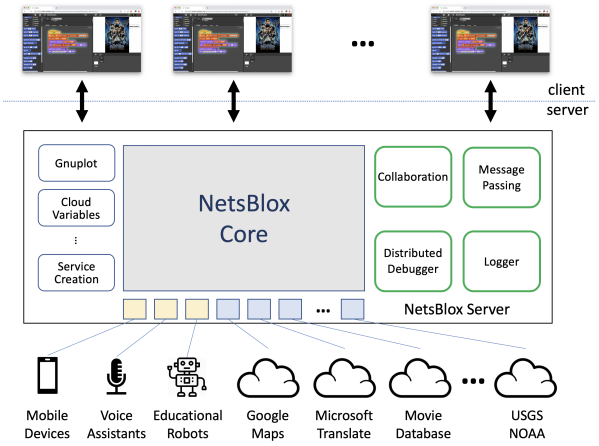

Fig. 8: NetsBlox architecture

It is important to note that the clients access the server via a well documented open RESTful API. Therefore, all the services and message passing support are available to potential alternative clients that do not need to be block-based. For example, we are already working on a Python front-end.

A. Make your own service

Up until recently, it was cumbersome for outsiders to add a new service to NetsBlox. One would either need to ask the authors to add something or implement it oneself and issue a pull request on GitHub. Many teachers are not JavaScript experts, so providing a service relevant specifically to their school was not really possible. To address this need, one of the latest NetsBlox services is called "Service Creation." It makes it possible to add one's own service that will then appear in the "call" block pull down menu under the Community and then *username* submenus.

Fig. 9: Create Your Own Service

To create the data field in Figure 9, user can drag in a csv file to turn into a variable (a two-dimensional array). The first row of this file should contain column headers (e.g., "date," "temperature," etc). The user can then pass the variable with the data as an input argument of the "createServiceFromTable" RPC as shown above. The new service will immediately become available with a set of predefined RPCs: one for each column of the table (e.g., "getTemperatureColumn", as well as one for each column as a function of the first column (e.g., getTemperatureByDate"). This latter option is useful when the data is some kind of time series and the first column contains the date/time values. The 'options' argument to ServiceCreation enables the user to change these default behaviors. It is even possible to provide a NetsBlox script for any of the desired RPCs. NetsBlox translates these into JavaScript and they become the code associated with the RPC.

This is yet another feature that helps students create relevant projects. Instead of emailing the students a data file, a teacher

can create a new service in a few minutes that becomes available to students instantly. If some revision needs to be made, the service can be updated just as easily, and all students will have the latest data automatically.

B. Alexa Integration

Enabling students to integrate voice assistants, like the Amazon Echo, into their distributed programs, is yet another opportunity to make programming more compelling and meaningful to young learners. Students can make games where players control their characters with voice commands or even control network-enabled robots using the RoboScape service from Section V. Even if they simply want to create a standalone skill, they are able to utilize many of the standard services such as Weather, MovieDB, and Translation.

NetsBlox provides an *Alexa* service, a collection of RPCs for creating a skill from a configuration (defined as a 2D list) along with additional helper RPCs. Using this service, users can define intents, give example utterances, and intent handlers as anonymous functions. Upon calling the creation RPC, the NetsBlox server creates the skill for the given user and stores the handlers in the database. When a command is spoken to the Alexa skill, the request is handled entirely by the NetsBlox server using the appropriate user-defined block-based intent handler. That is, when an intent is received by NetsBlox, the user-defined intent handler is retrieved from the database, compiled to JavaScript, and executed with the received values for each slot. As the intent handlers can utilize the message passing blocks, they can be used to forward messages to student projects, such as games where the players are controlled via Alexa.

IX. Evaluation

We have conducted several small-scale evaluation studies of NetsBlox through summer camps and in after-school settings. Since we cannot assume prior programming knowledge, the first two days of a week-long camp usually focus on introductory programming before tackling the more advanced topics NetsBlox was designed to teach.

Nevertheless, our studies have shown both statistically significant learning gains and increased student interest and engagement. Broll et al. report results from two summer camps that demonstrated between 15 and 20 percentage point gains in both CT and networking knowledge using a pre- and post-test [21]. Four summer camps with 62 students total were conducted in 2018 and 2019 focusing on robotics and cybersecurity using physical robots [16], [22]. Significant learning gains were achieved in both CT and cybersecurity. A quote from a participating high school teacher illustrates the level of student engagement: *"I did not see them on cell phones, they were engaged with programming their robot."* Feedback after a professional development workshop involving Distributed Computing using NetsBlox revealed teachers' ease with using RPCs and message passing blocks, excitement about how these features could expand students' projects to include various data sources from the internet, and an interest in using it in various ways in their schools–as part of teaching CS topics such as networks in AP CS Principles or in after-school camps [23]. NetsBlox has also been used to introduce first-year college students to programming during the first two weeks of an introductory programming course at Vanderbilt for several years now. The course teaches programming with MATLAB to non-CS engineering students. Anonymous surveys indicate that students with previous programming experience would rather not spend time with block-based programming, but most students appreciate the gentle introduction before switching to the main text-based language of the course.

The curricula used in these studies are all project-based. Many times we present a starter project, e.g., a current weather app or a chat program, and then let students work on enhancing it any way they like. For example, one student team in one of the camps added their own encryption algorithm to the chat project, so that they could keep their conversation private. Most camps conclude with an individual or team project of the students' own choosing. Innovative examples include various multi-player games such as a "Tron" clone, an interactive map interface for learning about country demographics and a running route planner on top of Google Maps.

The various collaboration models supported by NetsBlox have also been examined in multiple studies. Zacharia et al. compared students working in driver-navigator or driver-driver pairs, showing that the driver-driver configuration did not have the perceived imbalance in student agency that driver-navigator did [24]. These results align with a 2020 study by Tsan, et al. on pair programming in 4th and 5th grade, comparing programming on one computer versus students both acting as drivers on their own devices [25]. Student interviews suggested that the one-computer condition helped them communicate more with their partner, and the two-computer condition was preferred, but that students struggled to coordinate the programming with their partner.

Lytle et al. compared three types of driver-driver NetsBlox collaboration in a middle school summer camp [26]. In this study, pairs of students worked on a series of four game-themed programming projects using different collaboration styles in NetsBlox: separate, together, and puzzle. "Separate" involved students programming two separate NetsBlox Roles to complete a pong game, with one student programming the Left Paddle role, and the other taking the Right—with no collaborative editing across Roles. "Together" involved two students in the same Role, with synchronous editing to create a single paddle for Brick Breaker. "Puzzle" involved collaborative programming of a basket sprite to collect falling fruit—but with the blocks partitioned so that each partner could only access half of them, requiring partners to talk to coordinate efforts. In a fourth game project, 16 of 24 pairs chose Puzzle while 8 chose Together collaboration, with many citing that working with complementary blocks was more fun and interesting. An overwhelming majority expressed interest in collaborative programming in the future, with 27 preferring Puzzle-style, 17 preferring Together, and only 4 preferring to work Separately on future projects. These

statistics demonstrate that students in middle grades 6-8, aged 11-14, appreciate collaboration while learning to program!

The virtual robotics platform and the PhoneIoT app are recent developments. We are running three camps with 28 students and a week-long PD with four high school teachers this summer. Early feedback indicates that students are motivated and are engaging with these new technologies.

X. Conclusions

There is a widespread perception among high school-aged students that block-based environments are toys and not real programming languages. There is only so much one can do in a closed environment. Contrast that with the typical teenager's phone or laptop where the power of the entire internet is one click away.

But it does not have to be this way. Leveraging modern web technologies and the affordances of block-based programming environments can enable young learners to create projects that matter to them–making programming more relevant, motivating, and interesting. Projects such as distributed multi-player games, a shared whiteboard, and interactive global maps with superimposed climate data place powerful, creative possibilities in young learner's hands. Both teachers and students from middle grades up can successfully program such projects using NetsBlox. Furthermore, such projects can democratize access to learning important modern computing concepts, such as distributed computing and computer networking. Until now, these topics have only been taught to computing undergraduates, despite their importance to computer literacy for everyone.

Let us summarize what's possible once we remove the walls around a block-based programming environment:

- Student programs can access the wealth of information and services available on the internet. This makes it possible to create all kinds of STEAM-related projects sparking the interest of students who are traditionally not attracted to computing. For example, Figure 10 shows a project visualizing up-to-date pandemic information anywhere in the world.
- Being able to create programs that can communicate with each other opens up a world of online multi-player games and social apps for students to create.
- A novel approach to robot and device programming becomes possible. This enables collaborative robotics, remote control of games and robots with mobile devices, voice assistant integration, and an engaging, hands-on way of teaching cybersecurity.
- Novel forms of collaboration, including truly remote teamwork and sharing of in-process work can be supported seamlessly. In the age of COVID-19, this is a crucial requirement.

As we have shown, a lot of added functionality becomes available once programs have access to the internet. The most important consideration is to keep the abstractions that provide this access simple and intuitive. When a new extension is provided to the typical block-based environment, it comes with many new blocks. This makes it difficult to learn them and find them, especially if multiple extensions are used. Instead, a more general mechanism should be provided. NetsBlox added just two new abstractions introducing three new blocks (*call*, *send*, *when I receive*) to those typical of block-based environments, to provide a wide array of new capabilities. Furthermore, these two new abstractions are similar to ones that many students are already familiar with: an RPC is like a custom block, and messages are similar to events. This makes them intuitive and easy to learn and use.

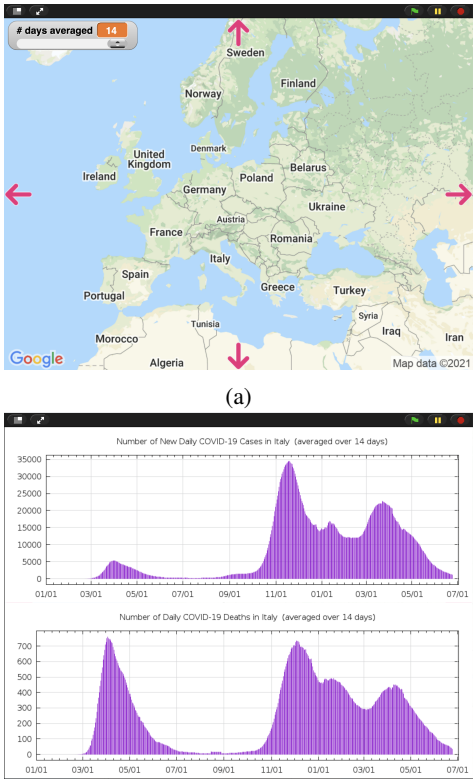

Fig. 10: Clicking on an interactive map of the world (a) shows up-to-date COVID data in the selected country (b).

Another important consideration is to keep the environment extensible, even by the users themselves. Adding services, including support for new devices, should not require new custom blocks or any changes to the client code or the interface. Moreover, users themselves can add their own online data services for everyone to use without leaving NetsBlox.

Once you show students the wide variety of advanced projects and technologies such an environment enables with just a few blocks of code, they will quickly reconsider the misconception that block-based programming is just for little kids. As two students said last summer: "It's really cool to see real world experience and real world data and real world things," and these projects help "a lot more people think [block-based programming] was really cool." A teacher added: "With the virtual delivery of my course, allowing students to collaborate in real time on a project, and understand HOW the collaboration works is a great learning experience...."

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1835874, the National Security Agency (H98230-18-D-0010) and the Computational Thinking and Learning Initiative of Vanderbilt University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] F. J. Rodríguez, K. M. Price, and K. E. Boyer, "Exploring the pair programming process: Characteristics of effective collaboration," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 507–512. [Online]. Available: https://doi.org/10.1145/3017680.3017748

[2] "NetsBlox website," https://netsblox.org, cited July 1, 2021.

[3] B. Broll, A. Lédeczi, P. Volgyesi, J. Sallai, M. Maroti, A. Carrillo, S. L. Weeden-Wright, C. Vanags, J. D. Swartz, and M. Lu, "A visual programming environment for learning distributed programming," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 81–86.

[4] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick, "Scratch: A sneak preview," in *Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing*, ser. C5 '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 104–109. [Online]. Available: http://dx.doi.org/10.1109/C5.2004.33

[5] B. Harvey, D. D. Garcia, T. Barnes, N. Titterton, O. Miller, D. Armendariz, J. McKinsey, Z. Machardy, E. Lemon, S. Morris, and J. Paley, "Snap! (build your own blocks)," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: ACM, 2014, pp. 749–749. [Online]. Available: http://doi.acm.org/10.1145/2538862.2539022

[6] A. Kelly, L. Finch, M. Bolles, and R. B. Shapiro, "BlockyTalky: New programmable tools to enable students' learning networks," *International Journal of Child-Computer Interaction*, vol. 18, pp. 8 – 18, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212868918300394

[7] S. C. Pokress and J. J. D. Veiga, "MIT App Inventor: Enabling Personal Mobile Computing," 2013.

[8] L. Moroney, "The firebase realtime database," in *The Definitive Guide to Firebase*. Springer, 2017, pp. 51–71.

[9] K. P. Birman, "Consistency in distributed systems," *Reliable Distributed Systems: Technologies, Web Services, and Applications*, pp. 375–390, 2005.

[10] B. B. Lim, C. Jong, and P. Mahatanankoon, "On integrating web services from the ground up into CS1/CS2," in *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 2005, pp. 241–245.

[11] L. Assunção and A. L. Osório, "Teaching web services using. NET platform," in *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, 2006, pp. 339–339.

[12] D. Garcia, B. Harvey, and T. Barnes, "The beauty and joy of computing," *ACM Inroads*, vol. 6, no. 4, pp. 71–79, 2015.

[13] D. Franklin, D. Weintrop, J. Palmer, M. Coenraad, M. Cobian, K. Beck, A. Rasmussen, S. Krause, M. White, M. Anaya, and Z. Crenshaw, "Scratch encore: The design and pilot of a culturally-relevant intermediate scratch curriculum," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 794–800. [Online]. Available: https://doi.org/10.1145/3328778.3366912

[14] S. J. Garber, "Searching for good science: the cancellation of NASA's SETI Program," *J Br Interplanet Soc*, vol. 52, pp. 3–12, 1999.

[15] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "SETI@home-massively distributed computing for SETI," *Computing in Science Engineering*, vol. 3, no. 1, pp. 78 –83, jan/feb 2001.

[16] Á. Lédeczi, M. Metelko, X. Koutsoukos, G. Biswas, M. Maróti, H. Zare, B. Yett, N. Hutchins, B. Broll, P. Völgyesi, M. B. Smith, and T. Darrah, "Teaching Cybersecurity with Networked Robots," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 2019, pp. 885–891.

[17] BirdBrain Technologies, "Remote Robots," https://www.birdbraintechnologies.com/remote-robots/, cited 2021 July 1.

[18] E. B. Witherspoon, R. M. Higashi, C. D. Schunn, E. C. Baehr, and R. Shoop, "Developing computational thinking through a virtual robotics programming curriculum," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, Oct. 2017. [Online]. Available: https://doi.org/10.1145/3104982

[19] B. Bennett, "Accurate distance calculation using gps while performing low speed activity," Master's thesis, University of Oregon, Jun 2018.

[20] S. Grover and R. Pea, "Computational thinking: A competency whose time has come," *Computer science education: Perspectives on teaching and learning in school*, vol. 19, 2018.

[21] B. Broll, Á. Lédeczi, H. Zare, D. N. Do, J. Sallai, P. Völgyesi, M. Maróti, L. Brown, and C. Vanags, "A visual programming environment for introducing distributed computing to secondary education," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 189–200, 2018.

[22] B. Yett, N. Hutchins, G. Stein, H. Zare, C. Snyder, G. Biswas, M. Metelko, and Á. Lédeczi, "A hands-on cybersecurity curriculum using a robotics platform," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 1040–1046.

[23] S. Grover, V. Cateté, T. Barnes, M. Hill, A. Ledeczi, and B. Broll, "First principles to design for online, synchronous high school cs teacher training and curriculum co-design," in *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, 2020, pp. 1–5.

[24] Z. Zacharia, D. Boulden, J. Vandenberg, J. Tsan, C. Lynch, E. Wiebe, and K. Boyer, "Collaborative talk across two pair-programming configurations," in *A Wide Lens: Combining Embodied, Enactive, Extended, and Embedded Learning in Collaborative Settings, 13th International Conference on Computer Supported Collaborative Learning (CSCL) 2019*, vol. 1, 2019.

[25] J. Tsan, J. Vandenberg, Z. Zakaria, J. B. Wiggins, A. R. Webber, A. Bradbury, C. Lynch, E. Wiebe, and K. E. Boyer, "A comparison of two pair programming configurations for upper elementary students," ser. SIGCSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 346–352. [Online]. Available: https://doi.org/10.1145/3328778.3366941

[26] N. Lytle, A. Milliken, V. Cateté, and T. Barnes, "Investigating different assignment designs to promote collaboration in block-based environments," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 832–838.