

Educational Robotics with a Twist*

Hamid Zare
Vanderbilt University
Nashville, TN, USA
hamid.zare@vanderbilt.edu

Miklos Maroti
Vanderbilt University
Nashville, TN, USA
mmaroti@gmail.com

Akos Ledeczi
Vanderbilt University
Nashville, TN, USA
akos.ledeczi@vanderbilt.edu

Abstract

We present a novel way to program wireless networked educational robots. RoboScape is a new service that we added to NetsBlox, a collaborative, blocks-based programming environment based on Snap! NetsBlox provides simple abstractions to create distributed programs. RoboScape uses these abstractions, Remote Procedure Calls (RPC) and message passing, to program the robot. The program runs in the browser and not on the robot. It remotely controls the robots with RPCs and the robot provides sensor values by sending messages back. This approach makes programming robots as easy as programming sprites.

Keywords visual programming, robotics, cyber-security, computer science education

1 Introduction

Computer networking and distributed computing concepts have become an important part of computer science and modern life in general. These technologies are everywhere, from social networks, through online multi-player games to self driving cars. The lack of educational tools to support effective teaching of key underlying concepts of distributed computing and computer networking to high school students prompted us to design and develop NetsBlox. NetsBlox extends Snap! with a few selected abstractions to allow for the creation of distributed applications including Remote Procedure Calls (RPC) and message passing.



Figure 1. "getMap" RPC from Google Maps service

RPCs provide access to free online data sources and services such as Google Maps, weather, movies or plotting. RPCs in NetsBlox are much more than just calls to web APIs; they are simpler to use and they keep state information. A number of related RPCs are grouped into services and logically separated as shown in figure 1. Some services even send messages to the user's program.

Message passing blocks are similar to events in Snap! with the addition that they carry data and can be sent to remote targets (other NetsBlox programs across the network/internet).

*This work was supported by the National Science Foundation under grant CNS-1644848 and the National Security Agency.



Figure 2. Message passing blocks in NetsBlox

These two abstractions open the door to teaching some of the fundamental concepts of distributed computing such as communication protocols, shared state, synchronization, addressing, latency, message loss, etc.

2 Robotics Education

Robots are an excellent platform to teach computer science and engineering in K12 because they are engaging and motivating. Traditionally educational robots are programmed using either a blocks-based or simple text-based language, the program is compiled and then downloaded to the robot. This adds unnecessary complexity, makes the development cycle cumbersome and hinders debugging. Wouldn't it be great if programming robots was as easy as programming sprites in Scratch or Snap!?

2.1 RoboScape

Leveraging the integration with NetsBlox, RoboScape provides the means to control robots and handle sensory feedback by utilizing existing abstractions in NetsBlox. The student's code executes entirely in the browser, hence, there is no need to download anything. It also allows the student to easily verify their logic by thinking through the problem as if they were programming the sprites to move on the stage. In addition, the students can take advantage of the debugging features that NetsBlox and Snap! provide such as stepping through the code, lively execution, variable inspection, and more.

RoboScape is implemented as a NetsBlox service meaning that it provides a collection of stateful RPC calls and takes advantage of messaging features. If desired, access to multiple robots can be granted to the students allowing them to control as many robots as they want from the same program.

The RoboScape service handles the communication between the users and the robots. This service includes calls to query a sensor as well as to control the robot behavior. Each call takes a robot ID as its first argument making targeting different or multiple robots as easy as changing a single id value. Our first version supports the Parallax ActivityBot 360. Some of the RPCs include:

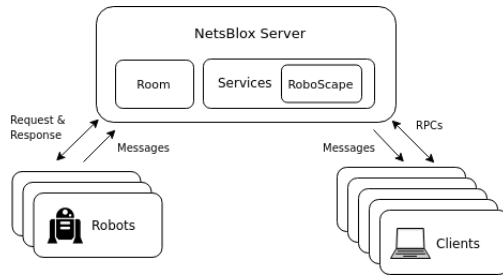


Figure 3. RoboScape architecture: multiple clients communicate with multiple robots through the NetsBlox server

- is alive: check whether a robot with a given ID is alive and responsive.
- beep: make the robot's buzzer module beep.
- set speed: set individual wheel rotation speed.
- get range: calls the ultrasonic range sensor module to probe and return the range.
- get ticks: returns the number of ticks for each wheel counted by the optical encoder.
- listen: subscribe to messages from the robot.

In addition to replying to user requests, the robots can also push messages to the server, some of which end up going to the clients. If the user wants to receive these messages they will first subscribe to receiving updates from the robot through the "listen" RPC and then create the message handlers for target messages. These include:

- identification message: sent every second and is used to ID the robot and keep the connection alive.
- range: whenever the "get range" RPC is called, a message with the sensor value is sent (in addition to being sent as the return value of the RPC call).
- button pressed: updates about individual button presses which can then be used to determine different button press patterns, e.g., short or long press.
- whiskers: informs the client every time the robot's whiskers are pushed or released and can be used to detect collision, obstacles, etc..

3 Bonus: Teaching Cybersecurity

We have also designed, implemented and tested a variant of RoboScape where most RPCs were replaced by a single one called *send*. The two arguments of *send* are the robot id and the textual command to send. This version of RoboScape uses only two message types: *robot command* and *robot message*. They have two fields: the robot id and the command (or message). The idea behind these changes is to support cybersecurity education. Robot commands, sent from a program to the robot, and messages, sent from the robot to the user, can be overheard by other users. Since they are sent in clear text, students can eavesdrop on each other and even hijack each others' robots. This is of course simulated: the



Figure 4. A partial RoboScape project implementing driving the robot with keyboard commands

NetsBlox server sends NetsBlox messages to all the clients that are listening to a given robot.

Students working with RoboScape quickly realize that they have to secure their robots. This provides an excellent entry point to introduce encryption. RoboScape supports Caesar's cipher. Initially, when the robots are booted they are not using any encryption. The clients then can issue a "set key" command to set up the encrypted communication that makes the robot and their program agree on the shift value. Eventually, they realize that key exchange should not happen in the clear either. The next step is to use the button on the robot that auto generates a key and displays it by toggling its LED. Students can write this down and type in their program.

Some more ambitious students can then progress to breaking the cipher using brute force. Other modules can include replay attacks and their prevention by utilizing sequence numbers and denial of service attacks and their remediation by rate limiting.

We have tested RoboScape and the cybersecurity curriculum with 24 students in grades 7-12 in a week-long summer camp. The engagement and the feedback received were overwhelmingly positive.