WebProto: Aggregate Programming for Everyone

IEEE Self-Adaptive and Self-Organizing Systems 2013 Demonstration

Kyle Usbeck
Raytheon BBN Technologies
Cambridge, MA, USA, 02138
Email: kusbeck@bbn.com

Jacob Beal
Raytheon BBN Technologies
Cambridge, MA, USA, 02138
Email: jakebeal@bbn.com

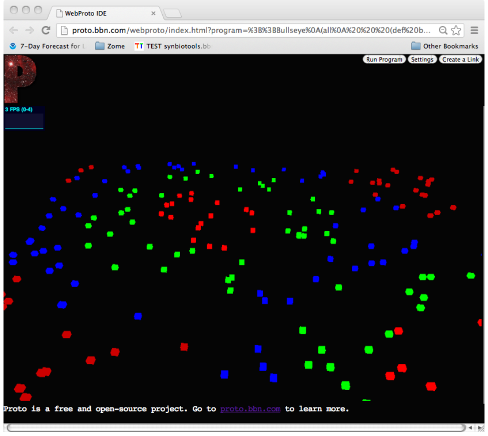
I. Goal: Make Aggregate Programming Accessible

Developing self-adaptive and self-organizing systems can be a difficult task, even for experts, because it has traditionally required programming individual devices in the hope that they exhibit the desired global behavior. A new and better approach has been emerging: program the global behavior and derive the local behaviors—*aggregate programming*. In recent years, there have been a number of key advances in aggregate programming for distributed systems, e.g., [1], [2], [3], [4], [5]. The tools associated with these projects have serious shortcomings, however, which typically include:

- **High barriers to entry.** Most tools require the user to install some piece of software (where the difficulty of installation and overall reliability varies highly on different platforms) and learn its particular usage patterns and set of quirks.
- **Disjoint components.** Most distributed algorithm tools focus on only one aspect (rarely two aspects): language, simulator, *or* deployment. It is challenging to construct a development environment that harmoniously integrates code-editing, simulation, and deployment.
- **Lack of active collaboration frameworks.** Sharing the distributed algorithms with students / colleagues / collaborators and reliably / efficiently showcasing their abilities is difficult in most existing toolkits.

Our aim is to make the state-of-the-art advances in aggregate programming readily available to a broad community of students, teachers, system programmers, and researchers by alleviating the shortcomings associated with current aggregate programing toolkits. This demonstration showcases our new WebProto software: a free and open-source web application that provides the Proto aggregate programming platform as a universally accessible code editor, compiler, and simulator with an easy-to-use collaboration framework.

WebProto allows any person with a modern browser to explore aggregate programming and self-organization and to easily develop novel software based on these approaches. We illustrate this capability with a web-based tutorial on Proto. This tutorial includes WebProto examples and exercises, also demonstrating how WebProto can be used as a simple platform for developing online curricula. Building on this demonstration, we hope to encourage broader adoption of aggregate programming in research, industry, and education.



Proto is a free and open-source project. Go to proto.bbn.com to learn more.

Fig. 1. Example of WebProto running a program that creates overlapping "bullseye" patterns around selected devices.

II. WebProto

WebProto is inspired by the example of NetLogo [6], a widely used educational tool for investigation of agent models and distributed systems. WebProto aims to provide similar easy access to the new aggregate-level programming models.

WebProto is built on top of MIT Proto[1], the current primary implementation and distribution of Proto. MIT Proto comprises four main components: (1) a library of Proto code for common self-organization primitives, (2) a compiler that turns Proto code into executable binaries, (3) the Delft virtual machine for executing binaries, and (4) a fast simulator, capable of testing scenarios with dozens to thousands of stationary or mobile devices.

The current distribution of MIT Proto has three main obstacles with respect to usability and accessibility. First and foremost, installation of MIT Proto depends on a complicated bootstrapping via GNU autotools, run by a sequence of console commands. This presents a formidable obstacle to the casual user and frequently encounters difficult to debug problems, particularly on Windows systems, which do not natively support the GNU toolchain. Second, program editing and simulation are disconnected workflows: programs are edited separately as text files, while simulations are invoked on the command line with a long sequence of configuration

[1] http://proto.bbn.com/

arguments. Finally, there is no intuitive interface for the simulator, which is controlled through key-bindings that can only be discovered in its (separate) PDF manual.

WebProto provides a unified interface that addresses all of these issues. In the conversion from command-line to web application, WebProto treats each of the four components differently. At the center of this conversion is a JavaScript implementation of the simulator, rebuilt from the ground up based on WebGL and the three.js library[2]. To ensure fidelity between implementations, the compiler and virtual machine (VM) are kept identical: the compiler is wrapped for execution as a CGI service, and the VM is compiled to a JavaScript implementation using emscripten[3]—note that this does impose a significant inefficiency, which is being addressed by an ongoing port of the VM to JavaScript. The self-organizing library code is untouched, and used by the compiler as usual.

The gap between editing and simulation is bridged by splitting the WebProto page between the 3D simulator display and an Ace code editor[4], configured for Proto code. Configuring and running simulations is done through a a settings pane and buttons at the top of the screen. When the user runs a simulation, their code is sent to the compiler web service, which returns a JSON script. The simulator then interprets that script into VM code and configures a network of devices with VM instances executing that code.

All of these components can be configured with extended URL arguments as well, such that they start in a desired configuration. Furthermore, WebProto enables both curriculum design and also collaboration between distributed-system developers with a “Create a Link” button. This button allows educators or developers to easily record and share programs and simulator settings by generating a single URL.

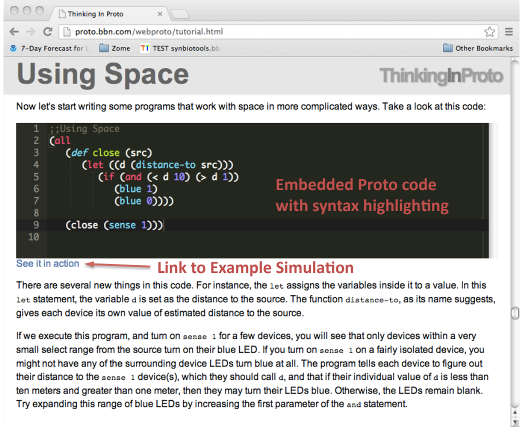
WebProto is publicly available two ways: first, we have set up an instance of WebProto, served at http://proto.bbn.com/webproto. This service can be used directly, embedded, or linked from other pages and services. Second, we have added WebProto as a new component of MIT Proto, available at http://proto.bbn.com/ as free software under an open license. Anyone is thus free to contribute improved code to WebProto and to set up their own instances.

Note also that the editor and compiler, although both used by the simulator, are set up as independent services so that they can be used separately as well. For example, the tutorial described in the next section embeds read-only instances of the Proto-configured code editor in order to display examples.

III. Demonstration: Thinking In Proto Tutorial

The main subject of our demonstration of WebProto is a tutorial entitled “Thinking in Proto.” This tutorial may be accessed online at: http://proto.bbn.com/webproto/tutorial.html

The document is an adaptation and update of the prior offline “Thinking in Proto” tutorial. It begins from the perspective of a student who knows little about programming,



Using Space

Now let's start writing some programs that work with space in more complicated ways. Take a look at this code:

```
;;Using Space
(all
  (def close (src)
    (let ((d (distance-to src)))
      (if (and (< d 10) (> d 1))
        (blue 1)
        (blue 0))))

  (close (sense 1)))
```

Embedded Proto code with syntax highlighting

See it in action

Link to Example Simulation

There are several new things in this code. For instance, the `let` assigns the variables inside it to a value. In this `let` statement, the variable `d` is set as the distance to the source. The function `distance-to`, as its name suggests, gives each device its own value of estimated distance to the source.

If we execute this program, and turn on `sense 1` for a few devices, you will see that only devices within a very small select range from the source turn on their blue LED. If you turn on `sense 1` on a fairly isolated device, you might not have any of the surrounding device LEDs turn blue at all. The program tells each device to figure out their distance to the `sense 1` device(s), which they should call `d`, and that if their individual value of `d` is less than ten meters and greater than one meter, then they may turn their LEDs blue. Otherwise, the LEDs remain blank. Try expanding this range of blue LEDs by increasing the first parameter of the `and` statement.

Fig. 2. Part of the “Thinking in Proto” web tutorial, showing embedded Proto code with automatic syntax highlighting and a link that opens a new page with an example WebProto simulation using that code.

and then one by one introduces key concepts of Proto and its continuous space/time approach to aggregate programming.

The online version of the tutorial now includes the following features based on WebProto:

- Examples link to WebProto simulations configured to run the example in a new window, so that students can explore aggregate-programming concepts and algorithms immediately as they read about them.
- Exercises encourage students to actively participate in the tutorial by writing code, thus improving comprehension.

Figure 2 shows a snapshot of the tutorial page including embedded code and a WebProto simulation link. The reader is strongly encouraged, however, to try the tutorial themselves.

This tutorial is a useful resource in and of itself, making it possible for people interested in Proto to start learning and experimenting immediately, without first having to go through a long and error-prone process of installation. More importantly, however, it is a demonstration of how our creation of WebProto can generally lower the barrier of entry for curriculum development, software engineering, and scientific investigation in the areas of aggregate programming and engineered self-organization.

References

[1] J. Beal, K. Usbeck, and B. Benyo, “On the evaluation of space-time functions,” *The Computer Journal*, 2012.

[2] J. Beal and R. Schantz, “A spatial computing approach to distributed algorithms,” in *45th Asilomar Conference on Signals, Systems, and Computers*, November 2010.

[3] J.-L. Giavitto, O. Michel, J. Cohen, and A. Spicher, “Computation in space and space in computation,” Universite d’Evry, LaMI, Evry, France, Tech. Rep. 103-2004, 2004.

[4] J. Fernandez-Marquez, G. Marzo Serugendo, S. Montagna, M. Viroli, and J. Arcos, “Description and composition of bio-inspired design patterns: a complete overview,” *Natural Computing*, vol. 12, no. 1, pp. 43–67, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11047-012-9324-y

[5] M. Viroli, “Engineering confluent computational fields: from functions to rewrite rules,” in *6th Spatial Computing Workshop*, May 2013.

[6] E. Sklar, “Netlogo, a multi-agent simulation environment,” *Artificial life*, vol. 13, no. 3, pp. 303–311, 2007.

[2] http://threejs.org/

[3] https://github.com/kripken/emscripten

[4] http://ace.ajax.org/