End User Live Programming Environment for Robotics

Jannik Laval*

Department of Computer Science-Software engineering, University of Lyon, France

Submission: April 30, 2018; Published: June 15, 2018

*Corresponding author: Jannik Laval, Department of Computer Science-Software engineering, University of Lyon, Mines Douai, IA, F-59508 Douai, France, Email: jannik.laval@mines-douai.fr

Abstract

Robots are more and more presents in our environment. Even if the manufacturers make robots enough intelligent to integrate every place by using SLAM for example, customers will need to specialize their robots to make them more familiar and more adapted to the need. For that, customers need to integrate new features that should be easily added to the robot. For now, the development of new features needs computer-programming knowledge.

In this paper, we propose an approach to avoid the gap between customer and software robotics development. We define a EUD (End-User Development) environment based on the visual programming environment Scratch, which has already proven in children learning computer science. We explain the interests of the environment and show two examples based on the Lego Mindstorms and on the Robosoft Kompai robot.

Keywords: End User; Live Programming; Robotics; Scratch; Mindstorms; Extensible language; LabView; Phratch

Abbreviations: EUD: End-User Development; LOB: Logic of Bricolage; ROS: Robot Operating System

Introduction

Mobile autonomous robots are progressively entering the mass market. From the same product line, they have the very same behavior. When the customer buy a robot, it is for a specific goal, for example a robot vacuum has to clean a whole floor. Then, when the goal is reached, the customer wants to customize the behavior of the robot. For example, a robot vacuum cleaner should clean up two times more the kitchen than the living room. For another customer, the expectation could be different. For example, a robot vacuum can become a beverage server as we can do with a Turtlebot. Each user has its own goals for its own domains. It makes that users need to develop their own software for robotics. A July 2011 Gartner report (“Citizen Developers Are Poised to Grow”) indicated that non-professional developers would build at least 25 percent of new business applications by 2014. We can estimate that it will be the same for robotics. There is no possibility for a customer to choose the behavior of a robot other than changing its internal software. Every customer does not have the knowledge of how to develop software and even less who know to integrate it into a robot.

Moreover, in robotics, the software complexity is growing up because of the increase of the number of sensors, motors, behaviors and features of the robots. The robots can be ground vehicles, flying vehicles, or boat. Then there are robots with wheels, or legs. The complexity of the robots makes the software more and more complex. It is difficult for an end-user to develop its own software on a robot.

At one side, the user should have the possibility to develop its own features. To make this possible, users should use a specific environment that needs low expertise. This kind of environment is named End-User Development (EUD) environment. EUD can be defined as a set of tools that allows non-professional developers to create, modify or extend a software artifact [1]. EUD is more and more used in the development of pure software applications. People with expertise in other domain than computer science use them to develop their own software application [2]. There are some key concepts that make a system EUD [3]: it needs a low threshold and a high ceiling. The low threshold means that the user interface should be enough simple to do something that work. The high ceiling means that the user can create a more complicated solution. Some approaches, like the Logic of Bricolage (LOB) [4], exists to help for the conception of EUD.

At the other side, 50% of robots failures of robot rescuers are due to poor human-robot interaction [5]. It is due to the teleoperation of robot with non-adapted interface. This problem is real for all robots. The author of this book insists on the fact that all robots in case of rescue, are teleoperated, even
ones with autonomy. It means that robots should be controlled as easiest as possible, mainly with the same application, to avoid learning the User Interface each time the customer uses a new robot. The conclusion of this fact is that the software for developing features should be easy to understand and enough extensible to be adaptable to many robots.

In this paper, we propose a EUD for robotics, named phratch. It is a EUD based on the Scratch visual development environment [6]. Scratch is considered as a “media-rich programming environment” [7]. In this environment, programming is performed by combining jigsaw puzzle pieces. It is well used for learning software programming for children. The assumption is that every user can use this software. So, phratch reuses this paradigm.

This paper is organized as follows. Section II discusses related work. Section III provides a vision of what can be a live programming environment for robotics. The phratch software model followed by a presentation of the user interface and how the environment can be used is explained in Section IV. Finally Section V shows two examples and Section VI discusses the possibilities of the environment and future work. Section VII concludes the paper.

Related Work

Using a visual programming language to control robots is a well-known practice [8]. It gives an understandable representation of the expected behavior of the robots. It allows a non-expert person to develop behavior for robots. For example, the Lego Mindstorms kit and Microsoft Robotics Studio use blocks and connections to describe a robot sequence of actions.

CREATE Lab Visual Programmer [9] is an educational software application developed for Arts & Bots educational program. It is a visual programming environment with a goal of low barriers to entry, available for young students and enabling complex robot behaviors. It provides a native support of robotics. The construction of behavior is made by assembling command blocks. The blocks are high-level states where in our approach, the blocks are low-level and based on control (as in Scratch). This software also provides live programming and live debugging.

RoboStudio [10] is a Visual Programming environment. It allows one to program the interactive behavior of service robots. The environment allows the developer to specify graphical interfaces that enable novice users to customize the behaviors of the robot to their needs.

Choregraphe\textsuperscript{1} is the graphical programming interface designed and developed by Aldebaran Robotics. It is the programming software that allows users create and edit movements and interactive behaviors for Nao. As for Lab-View, the representation is a flow chart based on node-link representation.

Gostai Studio\textsuperscript{2} is a graphical programming interface for creating behaviors for robot or complex system. It works on top of Urbi and allows controlling all robots working on this language. The graphical view is a node-link representation that represents finite state machine.

Lego Mindstorms Ev3 and its predecessors, NXT-G and RoboLab [11], are based on LabView. Compared to the preview language, the software is compiled and executed on the robot. This is efficient, but has the disadvantage to not have any feedback from the robot. It means that a developer cannot see in real time the values of the sensors outside the robot.

The Puzzle framework [12] is a EUD environment dedicated to mobile application.

It is not directly related to robotics, but this work is strongly related to ours on two points: first, it uses jigsaw pieces to build applications. Second, they try to build a framework that is extendable as much as possible. The main difference is the support. Puzzle is made for developing mobile applications on a mobile. They don’t have to manage the networking data and real-time information that have to transit between the computer and the linked material.

Physical Etoys\textsuperscript{3} is a visual programming language that allows one to develop on top of multiple hardware like Mindstorms or Arduino with a visual interface. This tool is lower level than other approach. It displays the real source code in a graphical user interface. This kind of environment is really interesting to learn software programming.

Scratch [6,13,14] is a language that is used in education to learn computer science by creating games and interactive multimedia supports using a kind of jigsaw puzzle pieces, named “blocks”. Programs are created by snapping blocks together. Each block represents a command that can control an action in a custom 2D character, called “sprite”. Each sprite can have its own behavior, composed with one or more sequences of blocks. It also allows one to manipulate data like variables. Scratch provides live programming and live debugging. This means that the environment gives feed-back to the user during execution, and the user can modify during the execution values or sequence of blocks.

\textsuperscript{1}http://www.aldebaran.com/

\textsuperscript{2}http://www.aldebaran.com/products/studio/gostai_studio/

\textsuperscript{3}http://tecnodacta.com.ar/gira/projects/physical-etoy/
An extension of Scratch, named Enchanting, exists to develop on Lego Mindstorms NXT robots. This extension is similar to Snaps in terms of usage. The main difference is that Enchanting is dedicated to Lego Mindstorms, whereas our environment is extendable to allow usage with other robots.

A Vision of Live Programming Environment for Robotics

In Disaster Robotics [5], Robin R. Murphy writes that one of the problems of robot terminations is the human error. In detail, this is not really directly due to a human error of command but it is due to outdated interfaces and protocols. It means that the robot operator cannot use easily a robot because the user interface is not appropriate. In this section we explain our vision of what could be an End-User programming environment dedicated to robotics. We define five requirements, detailed in this section.

A programming environment for end user

A programming environment is basically austere because it should have numerous features, an access to all configurations, or a lot of shortcuts that makes it difficult to use by a new developer. In case of End-User Environment, the interface should be simple but not simplistic. It means that the user interface should be easy as possible to understand. It also should contain all the necessary features to make the user develop more advanced software applications for robotics.

Requirement 1: End user functionalities: An End-User Environment should have low barriers to entry. It must minimize the amount of time required to learn how to create programs. The user should understand the interface and should have a low learning curve to make a robot move. The environment should encourage creativity and personal adaptation of technology instead of learning a programming language. The use of End-User bricks was already experimented and validated by other visual programming languages, as Scratch or LabView. This requirement is partly solved. Then, the user should be capable to see and learn the language behind the bricks. An access to this feature is required to allow a user to make his competency grow and to use more advanced features.

A flexible environment

An environment should be flexible in term of extend-ability and in term of complexity. The extendability is the ability to connect new robots to the environment. A user expects to not learn a new environment for each robot he buys. It means that one single environment should accept to add new robots. The complexity is the ability for the user to create more advanced algorithms, by writing source code for example. The user interface should have an entry point to see the real source code and to allow modifying it.

Requirement 2: Pluggable to Robots: Each robot has its own protocol, its own behavior. The EUD should provide tools to help developers to connect new robots to make them usable in the EUD. Unlike the other requirements, this one is dedicated to the developer instead of user. Creating a feature, or a block that have to appear in the EUD should be the easiest as possible.

Requirement 3: Extensible language: To develop more complex algorithms, an advanced user needs to access to the source code and all the tools from a development environment, like a debugger.

A Live environment

To reflect a robot behavior, the environment should be dynamic. This means that the environment should provide and display the current status of the robot, mainly, the values of the sensors. Consequently, a user can expect to live program the robot: he could try and understand how the robot react to a specific command or a part of the current algorithm regardless of the main created algorithm. In this case, the environment is directly connected to the robot. It receives and sends information in live. For example, a user can see what are the values and the unit of the sensors during the development. This information allows one to understand and validate the expected values and the expected behavior related on.

Requirement 4: Live environment: The environment requires a live system that is capable to send atomic commands to the robot and to ask for current sensor values that can be provided to the user. These features should be available during the creation of the algorithm.

A Programming Environment for Robotics

The environment should have the same features as a standard programming environment. Moreover, it should provide features for controlling the robots. For example, it should allow the developer to stop the robot in case of emergency. The environment should also contain tools to help users, for example, when they are debugging a robot behavior.

Requirement 5: Programming environment for robotics: The environment should provide all the necessary tools to command a robot or to introspect a robot. The environment has to be safe. For example, when sending a command to a robot, an error can occur. In a developing environment, a debug window will appear and display the error. When we are using robots, the environment should ensure that the robot is stopped. There are two possibilities: in case of beginner user, the debugger should not be displayed and the algorithm continues to run. In case of advanced user, the debugger window appears and the robot stops, waiting for a new instruction.

http://enchanting.robotclub.ab.ca/tiki-index.php

How to cite this article: Jannik L. End User Live Programming Environment for Robotics. Robot Autom Eng J. 2018; 3(2): 555608.
DOI: 10.19080/RAEJ.2018.03.555608.
Phratch Software Implementation

To meet all the requirements presented in the previous Section, we decided to implement a new EUD by extending the Scratch software application. Scratch interface has already proved its efficiency for learning computer science to children or students [7,6]. Extending the jigsaw block paradigm responds to the requirement about “End-User functionalities”.

However, it needs to be changed to respond to the other requirements: we have made it extensible, modular and accessible for users and developers. In this section, we explain the new features implemented in phratch to respond to the requirements.

Architecture

An overview of the system architecture is shown in Figure 1 Phratch is on top of a full software environment named Pharo [15], which allows developers to create API for robot control. The created API can be developed directly inside Pharo, or it can be an external call to a library if the API is developed in another language. The Pharo environment is based on a virtual machine that allows the language independent of the hardware.

User Interface

The user interface is presented in Figure 2. It has the same feeling as Scratch: on the left part, there are categories of blocks with, for each, a list of available blocks. In phratch we can build specific categories for the blocks of a robot. In Figure 2, the categories “Pure” and “Ev3” represents the available features for the robots Kompai and Mindstorms. At the center of the screen there is the pane where the user can build and try algorithms. In the figure, there is a simple script that connects to the robot, start it and stop it if it detects an obstacle. On the right, there are a basic simulation interface and the list of the robots. In the figure, two robots are available. An important difference is that in Scratch each element is called a sprite, it is represented by a picture and can be controlled in the simulation space. In phratch, an element can be a robot. It is represented on the screen identically as a sprite. It provides dedicated blocks for controlling robots.

Watchers

A watcher is a small box on the screen that shows at real time a value. In Scratch, watchers exist to show variables. In phratch, we extended the behavior to make it available for sensors of the robots. Each watcher is refreshed constantly to show the current state of the robot. In Figure 2, two watcher are displayed, one to see if the robot is connected, and one to see the current motor speed. The robot is not connected, consequently the speed of the motor is not known.
Live editing

As expected when programing a robot, a user needs to control the robot during its execution. It means that when an algorithm is running a user can modify it at runtime. During the execution of a group of blocks, the user can add new blocks in a sequence of blocks during the execution, it can also execute another block independently of the current execution.

Building blocks

A jigsaw block is a part of algorithm that finally sends one or multiple messages to a robot. To build a new block for a robot, the developer needs to create a subclass of the abstract class PhratchSpriteMorph. Then, in this new class, the developer has to create the methods to control the robot. If the API already exists, a simple call to the correct API is enough. Finally, to declare the method as a block into phratch, the developer declares a pragma on top of the method source code. The pragma has the following form:

```smalltalk
<phratchItem: ''
  kind: #CommandBlockMorph category: '' defaultValues:
    #() subCategory: #a
  special: #()>
```

a) phratchItem contains the label of the block.
b) kind is the kind of block. By default, the block is a CommandBlockMorph. Multiple other blocks exist like ReporterBlocks that represents a block returning a value.
c) category is the name of the category where the block will be available.
d) defaultValues is an array with the default parameters.
e) subCategory is a symbol that allows you to order blocks inside the category.
f) special is an array with special behavior that should be executed on the block before the block execution.

An example of a declaration of block is presented in Figure 3. The language is Smalltalk [16]. The first line is the signature of the method, with two parameters (motor and a Number). The second line is a comment that is used to generate documentation. The third and fourth lines are the pragma to declare the block. It needs two parameters (one motor and one number) that match with the method signature. The last line is the source code. Here there is only a call to an existing API.
Examples

Lego Mindstorms Ev3

Specifications: The Lego Mindstorms is a platform to learn robotics and understand the problematic about synchronization of sensors/actuators. It offers hardware modularity and conception freedom. The system is composed by a central unit, with a screen, buttons, a speaker, 4 output ports and 4 input ports. On this unit, we can plug four sensors and four motors.

To make phratch work with Lego Mindstorms, we use a library, JetStorm [17] we developed for teaching robotics to engineer students. This library is on top of a wifi bridge between our computer and the robots.

Blocks available in phratch: Multiple blocks provide the sensor type, status, and values. For the motors, some blocks provide the step number, the current speed and the current power. Then to move the robot, several blocks allow parameterizing motors, to synchronize motors and to start them. Other blocks are available to control the LEDs and the screen of the robot.

Labyrinth solver: The labyrinth was an exercise given to student in an engineer school. The students have no clue about computer science or robotics. The course is an introduction to robotics. We used phratch and Lego Mindstorms to practice. Figure 4 is a result proposed by a student. This result could probably be optimized, which is not the goal here. It shows that using phratch, students can learn robotics without learning a new language.
Kompai Pure

Specifications: The robot named Kompai, from the Robosoft company is 2 wheels differential drive robot. It is weighting 30kg and the maximum speed is 1m/s. It has multiple sensors: 1 laser, 16 infrared sensors, 9 ultra sonic sensors, 2 webcams, and 2 wheel rotary encoders for odometry. We wrapped the proprietary firmware, named Pure, into Pharo to communicate with the robot.

Blocks available in phratch: We have implemented the main features. Blocks returning the sensor values for each sensor and blocks returning the odometry values are available. To move the robot, we can send differential drive commands or step commands. We also implemented the block to control the trajectory service that allows declaring a list of segments that the robot follows.

Labyrinth solver: The simplest way to solve a Labyrinth is to follow a wall. As a first experimentation, the Figure 5 shows the algorithm to make the robot solve the labyrinth. It shows that with only few blocks it is possible for an end-user to define a robot behavior.

Discussion and Future Work

Requirements

The solution we propose solve partly the requirements enumerated in Section III.

For Requirement 1, about End User functionality, we choose to use the Scratch software and to extend it. Scratch is already used for learning Computer Science to children. It means that the paradigm used in Scratch can be a good solution for end-users. We extended it to make it work on robotics without changing the user interface.

Requirement 2, about the possibility to connect to robot without too much development, is partly solved. If an API is identified, the developer has only to create for each function a block, declared with a pragma. We plan for future work to make this declaration automatic. The automation will need the method signature and will generate the block automatically.

Requirement 3, about the need of an extensible language, is solved. In the user interface, it is possible to open a source code editor to create new behaviors. Different types of user can be identified and used. For now, phratch has two kinds of user: basic user and developer. We plan to extend it with at least an advanced user.

Requirement 4, about live programming, is solved. One of the important points of Scratch is the possibility to interact with the environment during execution. When we decided to extend Scratch, this feature was one of the most interesting to keep for robotics.

Requirement 5, about a programming environment for robotics, is partly solved. We expect to have an environment that interacts with the robots. Phratch allows one to develop when the robot is connected. The point that needs to be solved is when the robot has problems, for example when it loses the connection. One of the solutions can be to push a snippet of code on top of the robot that is executed in case of emergency. This snippet could be a future feature of phratch.

Future work

We plan to connect phratch on top of Robot Operating System (ROS). It will allow us to connect phratch on the ROS ecosystem and manipulate the entire robot available on top of it. The strong challenge is to make it easy to use for end-user.
Another future work is to compare Phratch to the existing software applications. We already listed the characteristic to compare: interactivity, genericity, extensibility, reusability, simplicity, performance, network disconnection tolerance, deployment of the source code inside the robot. This comparison needs to investigate the applications in terms of number of features and in terms of usage. This second point will need to build an empirical study built with users.

**Conclusion**

This paper proposes to discuss the requirements of an end-user development environment. Robotics is becoming more popular, we can consider that each user will create its own robots behavior. For that, the user should not learn a new language but need an interface that allows him to create easily its own robot behaviors. We list five requirements for EUD and we propose our own End User Live Environment for Robotics. It answers the listed requirements. Future work is mainly to make our work usable with ROS middleware.

**References**


**Your next submission with Juniper Publishers will reach you the below assets**

- Quality Editorial service
- Swift Peer Review
- Reprints availability
- E-prints Service
- Manuscript Podcast for convenient understanding
- Global attainment for your research
- Manuscript accessibility in different formats

(Pdf, E-pub, Full Text, Audio)

Unceasing customer service

Track the below URL for one-step submission

https://juniperpublishers.com/online-submission.php

008

How to cite this article: Jannik L. End User Live Programming Environment for Robotics. Robot Autom Eng J. 2018; 3(2): 555608.

DOI: 10.19080/RAEJ.2018.03.555608.