A Visual Environment for Reactive Robot Programming of Macro-level Behaviors

Floris Erich, Masakazu Hirokawa, and Kenji Suzuki

University of Tsukuba
Artificial Intelligence Laboratory
1–1–1 Tennodai, Tsukuba, Ibaraki, 305–8577 Japan
erich@ai.iit.tsukuba.ac.jp, hirokawa_m@ieee.org, kenji@ieee.org

Abstract. End-User Programming tools for robots typically allow end-users to combine macrobehaviors to design the behavior of a robot. Experienced developers are required to write these macrobehaviors. We present an approach for programming robots called Reactive Robot Programming (RRP). RRP allows end-users to construct macrobehaviors with minimal support from experienced developers by using uniform operators and clear separation between sensing, processing and actuation. The implemented approach consists of a Visual Programming Environment (VPE) and a robot architecture. In this paper we describe the VPE and robot architecture as well as present results from a pilot user study.

Keywords: End-User Programming, Visual Programming Environment, Reactive Robot Programming, Socially Assistive Robotics

1 Introduction

The field of Socially Assistive Robotics could benefit from using humanoid robots in therapy sessions [1]. One robot which is used in Autism Spectrum Disorder research is Aldebaran NAO [2]. Robot programming is concurrent and asynchronous and is hence complex [3]. Various tools have been developed to make it more accessible for developers to program robots, such as Choregraphe [4] and Microsoft Robotics Studio [5].

Choregraphe allows developers to write applications for robots ranging from the very simple to the very complex by combining macro-level behaviors (macrobehaviors) [4]. These macrobehaviors are typically part of a library. Tools like Choregraphe offer functionality for expanding the default library, by for example writing modules in a programming language and by constructing subgraphs. The subgraph approach allows users to create macrobehaviors by combining existing behaviors or by using the memory model of the robot for storing and retrieving data. It requires the user to have a good grasp of the memory model of the robot, and even then often requires knowledge of Python to create useful behaviors. The programming approach requires users to make a large step from using a specialized tool to using a general purpose programming language.

We propose an approach for constructing macrobehaviors which is more powerful yet more easy to use than subgraphs in Choregraphe. We call this approach Reactive Robot Programming (RRP). In this approach the user models the macrobehavior by describing the sensors to use, performing data processing on these sensors using operators and finally sending output to actuators. The macrobehaviors can either be run standalone on a robot, or be embedded in a robot programming framework such as Targets-Drives-Means [6] or Choregraphe. In this paper we describe a standalone implementation of this approach.

Unique about our approach is the usage of high level operators which can be customized using (procedural) parameters. Procedural parameters allow the user to separate between what an operator is supposed to do and how this should be accomplished. For example, the map operator takes a function that maps inputs to outputs as parameter. Procedures use only a subset of Python and are automatically embedded in running programs. We introduced this approach to programming robots in an earlier paper [7]. In this paper we introduce the Visual Programming Environment that we created for programming robots using RRP.

The rest of this paper is structured as follows: In Section 2 we review existing solutions for end-user programming. In Section 3 we elaborate further on our proposed solution by dividing it up into a technique, a tool and an architecture. In Section 4 we show a validation of our solution by means of a case study, a comparative analysis through a user study and a subjective assessment using NASA-TLX. In Section 5 we discuss limitations of our study and how to overcome these. In Section 6 we conclude the paper.

2 Literature review

Creating robotic systems that operate autonomous in complex environments has been a major goal of Artificial Intelligence research and has been previously explored by Minsky in the form of the Society of Mind [8], Arkin in the form of as Behavior-Based Robotics [9] and Brooks in the form of the subsumption architecture [10]. Our work is inspired by these explorations and can be used as part of systems that try to implement concepts from these works. The term *Reactive Robot Programming* has a loose relationship with the term *reactive planning* in the sense that Reactive Robot Programming is a technique that can be used for performing reactive planning.

Existing flow based programming tools such as Choregraphe and Microsoft Robotics Studio (MSRS) use embedded Visual Programming Languages in which information flows between blocks using connectors. In Choregraphe the blocks are called boxes, in MSRS they are called activities. Boxes in Choregraphe can read data, do computations and output data. In MSRS there are specialized activities to read data, do computations and output data. Due to this we consider RRP to be more closely related to MSRS than Choregraphe[1]. Neither Chore-

[1] Due to the inactivity of the Microsoft Robotics Studio project and the popularity of Choregraphe in the SAR domain we however decided to compare RRP with Choregraphe in our experiments.

graphe or MSRS support procedural parameters, which greatly increases the expressive power of a visual programming tool.

Another category of programming tools aimed towards making it easier for developers to create robot applications is component based systems, in which components are wrapped in a visual language. Examples of this approach are TiViPE [11] and RT-Middleware [12]. TiViPE aims to make it easy to add modules or data structures and documentation to the programming environment. The approach relies on the availability of the modules in some other language such as Java or C++. RT-Middleware is a Object Management Group standard for constructing middleware for robot technology. It uses CORBA and is hence suitable for creating distributed robotic systems. An open source implementation exists in the form of OpenRTM-aist. Just like for TiViPE, OpenRTM-aist aims to wrap existing components for use in a visual programming environment (in the case of OpenRTM-aist there are two tools for this purpose: RTSystemEditor and RTC Builder). Just like TiViPE and RT-Middleware, the RRP-VPE relies on the availability on some existing modules. The RRP-VPE currently does not provide an easy way of wrapping these. It however does allow the creation of intermediate nodes for data processing, whereas in TiViPE and RT-Middleware every node requires a dedicated component. The ability to easily wrap new sensor and actuator components would be a beneficial addition to the RRP-VPE.

Whereas flow based programming and component based systems both require the user to manually connect components, declarative systems for programming robots try to infer the structure of a program based on a specification declared by the developer. A recent example of such a system is Targets-Drives-Means (TDM) [6]. TDM applications consist of a set of behaviors which a robot can perform (such as walk towards an object or greet a person). For each behavior a schema is defined which detects an object in the environment (such as a human or a toy). A set of score calculators (such as distance to an object) determine which behaviors take presedence in the case that there are multiple schemes active. TDM manages the state of a robotic system and its evolution. Experiments have shown that TDM has a shorter learning curve than alternatives such as flowcharts [13] and that for end-users it is beneficial to use a graphical interface for writing applications [6]. TDM has the same limitation as component based systems such as TiViPE and RT-Middleware, namely that it requires components to be written using a different tool before they can be used in robot programming environment. In the case of TDM, RRP can be used to write functions, conditions and score calculators.

3 Solution

3.1 Approach

Our basic hypothesis is that end-users can construct complex behaviors if they separate sensing, processing and actuation and are given a visual environment to develop behaviors in. RRP is a platform for developing behaviors according to this. On a robot such as NAO the sensors and actuators are fixed and can

hence be supplied as a library. How the developer connects the sensors with the actuators depends on the actual use case of the developer. Instead of trying to predict exactly what the developer wants to accomplish, we offer a set of high level operators which can be used to perform the data processing. The developer can then use these operators for constructing an application. The developer models applications from top to bottom, starting with the actuator streams, which are connected using operators to intermediary streams, which are finally connected using the subscribe operator to actuator streams.

Our solution currently supports the following operators:

- One to one operations
  - **Subscribe** Sends inputs to actuators. Except the subscribe operator, all operators should be side effect free, hence subscribe is the only operator which enables the robot to create output.
  - **Timestamp** Adds a timestamp to values from the input stream. This is useful for combining data from various streams.
  - **Sample (with rate parameter)** Samples the input stream. At an interval specified by rate, checks if there is a new value available and if so outputs the value. This is useful to avoid doing unnecessary computations, such as sending messages to an actuator at a higher rate than supported by the actuator.
  - **ForgetAfter (with interval parameter)** Forgets values that are older than the defined interval. Always directly outputs its input. After a time period specified by interval has passed while no new value has been received, outputs a sentinel value (for example the value None if the runtime environment is Python). This is useful when combining messages from multiple streams while avoiding using old data.
  - **Filter (with predicate parameter)** Filters values based on a predicate procedure. If the predicate holds, the input is outputted, else it is ignored. This is useful when we want to apply some condition to the messages or for separating messages into different streams.
  - **Map (with transformation parameter)** Maps values based on a transformation procedure. The procedure is applied to each input value and the resulting value is outputted. This is useful for changing the format of input messages into a desired output format, such as angles in degrees to angles in radians or position data in one reference frame into another reference frame.
- Many to one operations
  - **Combine (with combinator parameter)** Combines values from input streams to a single output stream using a combinator procedure. The procedure takes as input the latest value of the input streams and outputs some combined value. This is useful when data is received by different sensors and has to be combined, such as data from a laser range finder and from the camera.
  - **Merge** Merges values from input streams to a single output stream. Always outputs the latest input stream changed. This is for example useful when multiple sensors are producing the same type of messages.

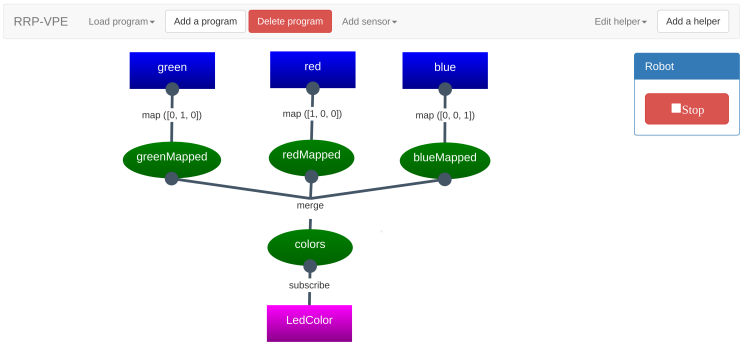

Fig. 1: A screenshot of the RRP-VPE. The top bar allows the user to add, delete and load programs and to add, edit and remove helpers. When a program is loaded the user can also add sensor streams using the top bar. The currently loaded application has three sensor streams (drawn as blue rectanges) that represent blob detectors for the colors green, red and blue. Map operators transform the inputs into an RGB intensity vector, putting the output in streams called greenMapped, redMapped and blueMapped (drawn as green ovals). The values are merged into a stream called colors. The subscribe operator takes the latest color and submits this to the LedColor actuator stream.

In an earlier paper we describe an elaborate example of how these operators are used in an actual application [7].

3.2 Visual Programming Environment

The Visual Programming Environment (VPE) is a web application which allows users to collaboratively develop applications for a robot. The front-end is implemented using the JavaScript framework JQuery, CSS framework Bootstrap, diagram library JSPlumb and data binding library Knockout. The back-end is implemented using the NodeJS runtime, the ExpressJS framework and the Neo4j Graph Database. Communication between front-end and back-end is realized using the asynchronous communication library Socket.IO. A screenshot of the VPE is included in Fig. 1.

A typical workflow of using the RRP-VPE is as follows: (1) Add a program, (2) add sensors, (3) add operators to process sensor data, (4) subscribe actuators to the processed data and finally (5) start the program from the VPE, which creates an instance of a runtime architecture connected to the robot.

3.3 Robot architecture

The robot architecture consists of various tools to run and debug programs on a robot. It was written in Python. There is a shell for running applications interactively, and there is a launch script for starting applications directly from the

command line. There is a division between the data model, database adapters and robot adapters, which allows the architecture to be used using different storage media (such as the graph database or an XML file) and different robots (currently only NAO is supported, in addition to some environments used for debugging). The engine is multi-threaded. Events raised by the robot are asynchronously send to the robot. Operators such as sample and forgetAfter maintain a timer thread.

4 Validation

As a preliminary validation we performed a pilot study with students and faculty members. All subjects had some experience with software development for robotics and/or Internet-of-Things appliances.

4.1 Comparative analysis

We performed a user study which compared developing macrobehaviors in the RRP-VPE with Choregraphe. We implemented four programs in both tools. We recruited four subjects to participate in the experiment: One master student, two PhD students and one faculty member. The subjects had a background in intelligent interaction technologies and were experienced in working with robots and with programming in their daily research. The subjects were hand selected to control for their level of experience with social robots. The subjects performed three tasks using each tool: Explaining the behavior of a program, finding a bug in a program and creating a program. For the explanation task we picked different programs for the subject to explain. For the debugging task we used a similar program in both tools, however the bug that we introduced was different for each tool. For the creation task the subject had to create the same application in both tools. We counter-balanced the experiment so that half of the subjects started with RRP-VPE and then used Choregraphe, and the other half started with Choregraphe and then used RRP-VPE. We also balanced the explanation task.

The programs which we used were as follows:

Explanation task A1: Mirror ball color Once a second the robot looks for balls of the color red, green or blue. If a ball with such a color is found then the robot changes its eye color to match the color of the ball.

Explanation task A2: Say ball color Once a second the robot looks for balls of the color red, green or blue. If a ball with such a color is found then the robot says the color of the ball.

Debugging task B: Distance to brightness The robot looks for a red ball. If found the robot calculates the distance to the ball. The robot changes the brightness of its eye LEDs based on the distance to the ball (full brightness if the ball is close, no brightness if the ball is far, a fraction if the ball is in between)

	Subject 1	Subject 2	Subject 3	Subject 4
Tool A	Choregraphe	RRP-VPE	Choregraphe	RRP-VPE
Tool B	RRP-VPE	Choregraphe	RRP-VPE	Choregraphe
Tool A explanation task	A1	A1	A2	A2
Tool B explanation task	A2	A2	A1	A1
Debugging task (both tools)	B	B	B	B
Creation task (both tools)	C	C	C	C

Fig. 2: Assignment of tools and tasks for each subject.

Tool	Task	S1	S2	S3	S4	Mean	SD
Choregraphe	A	00:52	02:19	02:22	02:20	01:58	41.7
RRP-VPE	A	01:16	01:14	01:15	01:32	01:22	5.4
Choregraphe	B	00:47	01:08	01:19	00:47	01:00	13.1
RRP-VPE	B	00:44	01:04	02:55	00:38	01:20	57.6
Choregraphe	C	*10:00*	*10:00*	*10:00*	*10:00*	10:00	0
RRP-VPE	C	04:31	02:42	05:10	05:15	04:25	71.10

Fig. 3: Duration per task per tool (in minutes). Each task was capped at 10 minutes. If a subject was out of time we emphasized the result. Standard deviation is in seconds.

Creation task C: Track close balls with arms, track far balls with head
The robot looks for a red ball. If found the robot calculates the distance to the ball. If the ball is close then the robot tracks the ball with its left arm. If the ball is far then the robot only tracks the ball with its head.

The dependent variables were the time taken for the task and task evaluation based on NASA-TLX. Random variables were age, experience with various programming constructs and gender. We tried to control for the variables of time and location by performing the experiments approximately at the same time of the day (around noon) and at the same location. Before starting each experiment the participant signed an ethical consent form.

The total duration of each experiment was at most 1.5 hours. The distribution of the tasks over the subjects can be found in Fig. 2. The duration per task per tool can be found in Fig. 3. A graph showing the mean duration and standard deviation can be found in Fig. 4.

4.2 Subjective assessment

We used Raw NASA-TLX for evaluating the total workload of using Choregraphe and our solution. The results can be found in graphical format in Fig. 5.

5 Discussion

The performed experiment shows that RRP-VPE outperforms Choregraphe for the creation task. Even though we only had a few participants this finding re-

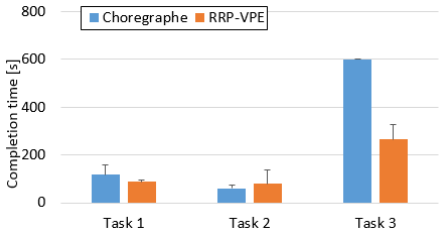

Fig. 4: Mean duration and standard deviation for each task and both tools

mains significant. For this experiment we recruited participants which have some experience with programming robots. A more realistic study would include participants which have little programming experience. Ideally we would perform this experiment with the actual target user of our research, namely therapists.

We suggest that applications modelled using the VPE are more simple than similar applications modelled using Choregraphe for three reasons. First, the responsibility for each kind of element is defined in RRP. Sensor streams read data and do not change the behavior of the robot. Ordinary streams are simple containers for data. Actuator streams change the behavior of the robot. Because of this separation we can make some assumptions such as that sensor boxes should be started automatically. In Choregraphe a box can have many responsibilities which also complicates the logic for starting and stopping a box. The second reason is that RRP has a graphical notation which tries to show all data. For example, parameters to operators are visible without having to use drilldown forms. Lambda functions are embedded into the graph, though the body of more complicated functions can be hidden by using a helper. Third, by using operators with procedural parameters we require the user to have less programming skill to be able to accomplish tasks. For example, to include simple Python expressions in Choregraphe the user needs to be familiar with writing functions and classes. For RRP-VPE the developer only needs to know how to write (multiline) Python expressions such as variable declaration/assignment and function calls.

Whereas our tool was shown beneficial for the creation task, the same was not true for the explanation and debugging task. The tasks were most likely too easy to complete and hence did not require the user to use any specific features offered by the platforms. For example, we allowed the user to assume the implementation of the boxes in Choregraphe as well as the helpers in RRP-VPE were correct. For the explanation task, if we would use non descriptive name for the boxes the participant would not be able to explain the behavior diagram so easily (having to look inside each box). For the debugging task, introducing a bug within a box or helper would make it significantly harder for the user to perform this task. At the same time we would greatly increase the chance of the user not being able to properly explain a behavior diagram or not able to find a bug within the allocated time.

6 Conclusion

In this paper we introduced the Reactive Robot Programming approach with an accompanying Visual Programming Environment. Through a pilot user study we showed that this solution outperforms Choregraphe for creating new macrobehaviors. Users are faster when creating macrobehaviors using RRP-VPE and feel a lower task load. In the future we want to repeat the experiment with end-users instead of experienced robot programmers.

References

1. Feil-Seifer, D. and Mataric, M.J., "Defining Socially Assistive Robotics", IEEE International Conference on Rehabilitation Robotics, June 28–July 1, 2005, Chicago, IL, USA, pp. 465–468.
2. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J. and Maisonnier, B., "Mechatronic design of NAO humanoid", IEEE International Conference on Robotics and Automation, Kobe, Japan, May 12–17, 2009.
3. Kortenkamp, D., Simmons, R. and Brugali, D., "Robotic Systems Architectures and Programming", in: Siciliano, B. and Khatib, O. (Eds.), "Springer Handbook of Robotics", Springer-Verlag, Berlin, Germany, 2016.
4. Pot, E., Monceaux, J., Gelin, R. and Maisonnier, B., "Choregraphe: a Graphical Tool for Humanoid Robot Programming", IEEE International Symposium on Robot and Human Interactive Communication, Toyama, Japan, September 27–October 2, 2009.
5. Morgan, S. "Programming Microsoft Robotics Studio", Microsoft Press, Redmond, USA, 2008.
6. Berenz, V. and Suzuki, K., "Targets-Drives-Means: A declarative approach to dynamic behavior specification with higher usability", Robotics and Autonomous Systems, Volume 62, Issue 4, pp. 545–555, 2014.
7. Erich, F. and Suzuki, K., "Cognitive Robot Programming using Procedural Parameters and Complex Event Processing", IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots, San Francisco, December 13–16, 2016.
8. Minsky, M., "The society of mind", Simon & Schuster, New York, USA, 1986.
9. Arkin, R.C., "Behavior-Based Robotics", MIT Press, Massachusetts, USA, 1998.
10. Brooks, R.A., "Intelligence without representation", Artificial Intelligence, Volume 47, pp. 139–159, 1991.
11. Lourens, T., "TiViPE — Tino's Visual Programming Environment", Annual International Computer Software and Applications Conference, Hong Kong, China, 28–30 September, 2014.
12. Ando, N., Suehiro, T. Kitagaki, K. and Kotoku, T., "RT-Middleware: Distributed Component Middleware for RT (Robot Technology)", IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Canada, August 2–6, 2005.
13. Berenz, V. and Suzuki, K., "Usability Benchmarks of the Targets-Drives-Means Robotic Architecture", IEEE-RAS International Conference on Humanoid Robots, Osaka, Japan, November 29–December 1, 2012.

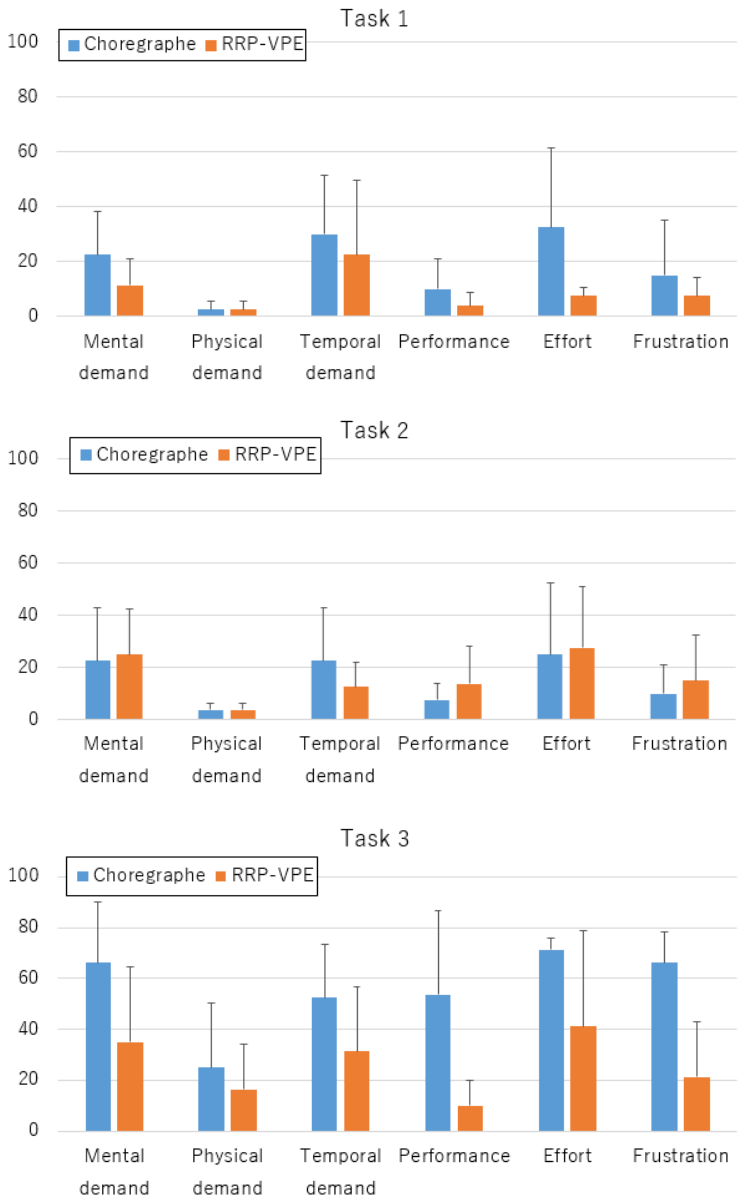

Fig. 5: Mean and standard deviation of NASA TLX evaluation for each task.