Visual Dataflow Language for Small Robots Programming

Grogorii Zimin
Mathematics and Mechanics Faculty,
SPbSU
Saint-Petersburg, Russia
Email: zimin.grigory@gmail.com

Dmitrii Mordvinov
Mathematics and Mechanics Faculty,
SPbSU
Saint-Petersburg, Russia
Email: mordvinov.dmitry@gmail.com

***Abstract*—REWRITE THIS SH.. The paper describes dataflow visual programming language based on DSM-approach. Its purpose is to be bridge between lightweight robotics languages for education and complex industrial languages. A short review of programming languages for robots is presented here. Also different approaches and architectures for developing control system for robots are considered. For demonstration of language usage, the paper provide solution for creating control system of robot based on subsumption architecture.**

1. Introduction

Programming languages for creating robotic controllers are actual topics of research oftenly discussed at major conferences, such as ICRA[1] or IROS[2]. Visual programming languages (VPLs) are also actively discussed for the last three decades, the largest conferences are held annually, e.g. VL/HCC[3]. VPLs are oftenly applied in robotics domain[4–8] allowing to create and visualize robotic controllers. Robotic VPLs are commonly used for educational purposes, making possible for students of even junior schools to create robotic programs. There already exists a great number of educational robotic programming environments based on VPLs, e.g. NXT-G[9], TRIK Studio[10], ROBOLAB[11], also there are some academic tools implementing interesting and novel approaches to educational robotics programming[4], [6], [8].

Robotic control programs have reactive nature: they transform data which is continuously coming from multiple sensors into the impulses on actuators. For this reason dataflow languages (DFLs) are well-suitable for robotics programming. Many researchers denoted the conveniency of dataflow visual programming languages (DFVPLs)[12], finding them more useful than textual DFLs, for example, because data flows explicitly displayed on the diagram. There are large and complex general-purpose and domain-specific development environments such as LabVIEW[13] and Simulink[14] that provide a large (and sometimes even cumbersome) set of libraries for robotics programming. More detailed discussion of robotics VPLs will be provided in section 2.

There is a large number of robotic constructor kits for learning the basics of robotics and cybernetics, such as LEGO MINDSTORMS[15], TRIK, ScratchDuino[**?**]. Modern programming languages that are used for programming those kits are based on the control flow model rather than on dataflow model. Control flow-based languages are good for solving scholar "toy" tasks, but may be inconvenient for programming more complex "real world" controllers that may be conveniently expresses on DFLs. The simple DFVPL may be considered as a useful step from educational VPLs to the programming languages that are used in unversities and industry.

In SPbSU cybernetic laboratory are conducted a number of studies aimed at improving the tools for design and programming of embedded systems for small robot platforms (e.g. LEGO, TRIK) using block diagrams[]. Goal of research paper is the development of novel extensible tool for programming all small robot platforms which discussed above in dataflow style. While dataflow approach is commonly used approach when each element is executing in separate thread or process. Our approach avoids it, because it brings a large overhead on target platforms, details presented in section 3. There are several commonly used robots controller architectures: Connell's Colony, Maes' Action-Selection, Arkin's Motor Schema, Rosenblatt's Distributed Architecture for Mobile Navigation, Brooks' Subsumption Architecture[16]. DFLs are suitable for Brooks' Subsumption Architecture, his approach are commonly used[4], [5], [7], [17]. So, our approach also uses Brooks Subsumption Architecture as the fundamental concept used in the our DFVPL.

The paper is organized as follows. An overview of robotics VPL and DFVPL is presented in section 2. A description of our language are proposed in section 3. Section 4 demonstrates typical robotic controller expressed in our language. Direction of future research works is given in section 6. Finally, the last section concludes the paper.

2. Overview and background

Robot programming environment can be divided into three categories: educational, which allows you to program small robots; industrial, which have a rich toolkit for creating robots control systems and different models; academic, which implement new interesting ideas, however, are often not available for download, or are not robust.

To educational, for example, can be referred EV3 Software development environment for the Lego Mindstorms EV3 kit, IDE NXT-G and ROBOLAB for LEGO MINDSTORMS NXT kit, TRIK kit and IDE TRIK Studio. These IDE make it easy to solve typical robot control tasks, e.g., find a way out of the maze, drive along the line using sensors by creation a primitive control system, which purpose is to teach the users the basics of programming and robot control. But their simplicity is bound with poor flexibility of the language. Actually these IDEs provide a consistent model of the robot control based on control flow model, that is described by visual graphical models.

In the industry there are very popular general-purpose IDE LabVIEW from National Instruments with the DFVPL G, and programming environment Simulink developed by MathWorks for modeling different dynamic models or control systems. These software products offer to the users a huge range of models and libraries to create any control systems, test benches, real-time systems, using model-driven approach. In particular, LabVIEW provides opportunity programming small robots. Although there is known of LabVIEW usage for educational purposes[18], most of the time in the educational process is spent on the study of the medium itself, not algorithms and robotic approaches. It should be noted that these environments are distributed under the commercial license.

Another example of an industrial system is the Microsoft Robotics Developer Studio (MSRDS)[19], which is free for academic use, and allow to easy create distributed robotic systems in terms of data flow, which components are represented as web services. MSRDS officially supports multiple robotic platforms, e.g., LEGO NXT[20], for which, however, It does not support offline mode. MSRDS has the ability to integrate with custom robotic platforms, but this environment is not supported in 2014.

There are a lot of research in this area, e.g., dissertation[4] describes a visual programming language in terms of extended machines Moore, [6],[7] describe visual $occam$-π program editor and tools $Transterpreter$, and its usage in swarm robotics control system. Article[8] describes DFVPL for beginners and environment, which provides a comfortable interface. But the technology has too weak functionality and requires significant improvements, and doesn't available for download.

Based on the overview it is clear that no one of these environments is not suitable for programming small robots in terms of data flows. At the same time closest to the desirable is TRIK Studio, as it is the only medium, which distributed with open source with a highly scalable architecture, that has the ability to extend itself by new VPL for robots and provides opportunity to reuse code base of "routine" operations such as the interaction with the robot.

3. Description of the language

Evolution of a domain-specific modeling (DSM) methodology allows to quickly create a fairly sophisticated visual programming languages[**?**].

4. Example

5. Implementation

The system is implemented as two plugins for TRIK Studio. The first one describes the visual language and provides visual editor for our system. It contains the metamodel of dataflow visual language and entirely generated by QReal DSM-platform. Plugged into TRIK Studio this module provides fully operational visual editor with all advantages of TRIK Studio control flow editor like modern-looking user interface, ability to create elements with mouse gestures, different appearances of links and so on. The time spent on the development of this plugin (not considering discussing and designing the prototype of visual language on paper) roughly equals three man-days. The benefit on exploiting the DSM-aproach is obviuos, the development of the similar editor from scratch would have been taken vastly more time.

The second plugin contains implementation of dataflow diagrams interpreter. Given the program drawn in editor (provided by first plugin) interpreter will transform that program into a sequence of the commands sent to a target robot (see fig. 1). The target robot can be one of the supported in TRIK Studio infrasctructure: Lego NXT or EV3 robot, TRIK robot, TRIK Studio 2D simulator or V-REP 3D simulator[21]. Commands are sent via high-level TRIK Studio devices API, a part of it presented at fig. 2.

The general acrhitecture of intepreter plugin is presented at fig. 3. Given dataflow diagram interpreter traverses, validates and prepares it for interpretation process. For each visited dataflow block implementation object is instantiated. Implementation objects are written in c++. Instantiation is performed by corresponding factory object. Implementation objects are then subscribed each to other like they are connected by flows on diagram, *publish-subscribe* pattern is used here. The set of initial blocks is determined next, those are blocks without incomming flows. After all that done preparation phase is complete and diagram starts beeing interpreted.

Interpretation process is not as straightforward as in most asynchronious dataflow environments. Usually components of dataflow diagram are executed concurrently, on differents threads, processes or even machines (that is actively exploited, for example, by Microsoft Robotics Developer Studio where dataflow diagram is deployed into a number of web-services). That is a pretty convenient way to invoke dataflow diagrams on a powerful hardware, but not a case when we talk about embedded devices. In out case we deal exactly with embedded devices (Lego NXT, EV3, TRIK, Arduino controllers), so we propose here another

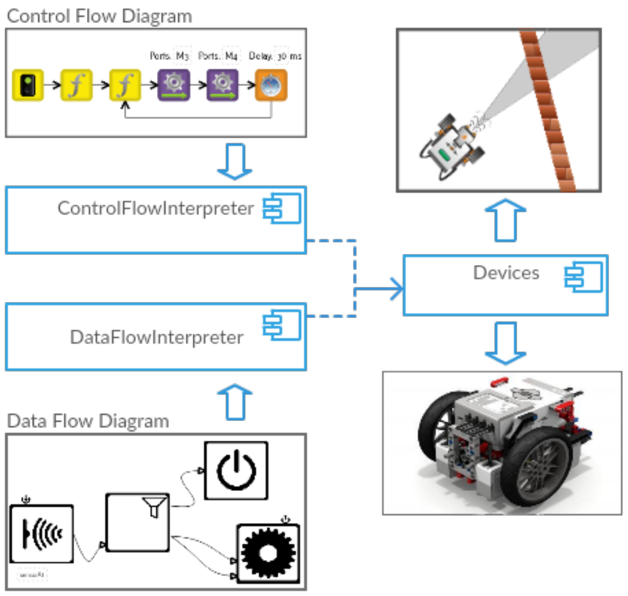

Figure 1: The general acrhitecture of the system

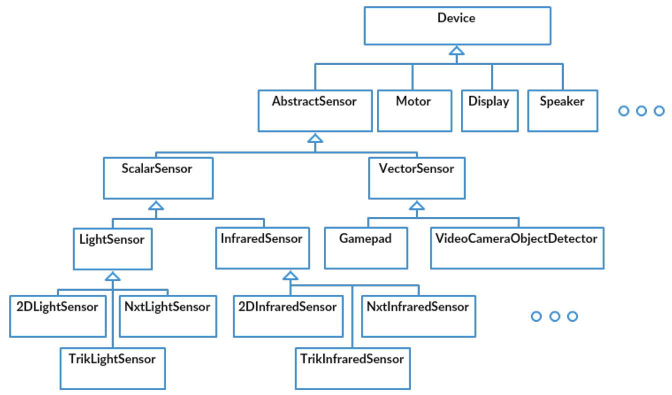

Figure 2: Partial architecture of devices used in dataflow interpreter

way of executing dataflow diagrams. The main idea is to intoduce global message queue and event loop for messages processing. When token is published by some block it is enqueued into messages queue and waits for its turn to be delivered to subscribers (fig. 4). In fact thus we *flatten* the execution, convert concurrent way of dataflow interpretation to a pseudo-concurrent one where we schedule invocation order on our own. It must be noted that this mechanism is similar to events propagation system of Qt framework. That is actively exploited in our implementation, where message processing is completely performed by $QEventLoop$ class and tokens delivering is done by Qt signal/slot system in $QueuedConnection$ mode.

Flat execution of dataflow diagram poses a number of small problems, one of them will be discussed here. Input device blocks (for example blocks publishing tokens from

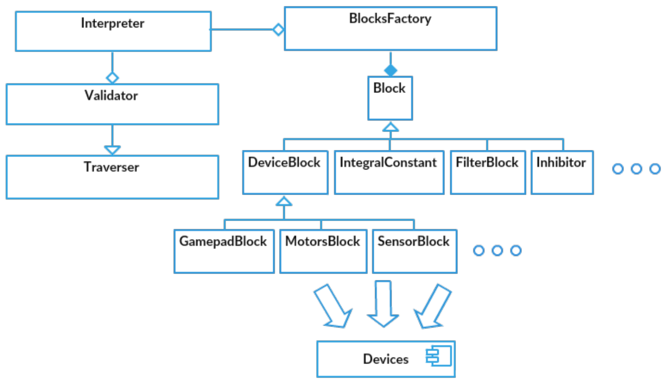

Figure 3: The general architecture of dataflow interpreter plugin

ultrasonic sensors) are constantly emitting tokens to subscribers. Subscribers transmit tokens to a next one (possibly in modified state) and so on. Thus there appears a chain of data processing. In out language that chain can activate control flow ports of blocks "reviving" them, so the control flow model is implicitly supported in out language (this is important in educational reasons). If later in this chain same input device block will be met then execution will come in a counterintuitive way. Such conflicts are ruled out with a simple heruistic that among all the blocks sharing one physical device only one can be active and that is the last activated one. Thus when the execution token comes into some device block it immediately "deactivates" conflicting ones. Other problems like messages balancing (in case when some block "flooding" the whole messages queue) will not be discussed here.

The last thing we should remark here is the presence of $Fork$ block in our language that usually is not provided by dataflow languages. Flattened model seems to work well on embedded devices, but sometimes users still need to use concurrent execution (for example for executing layers in subsumption architecture). For that reason $Fork$ block is introduced, it forks the execution into a number of platform-specific execution units (for example $pthreads$ on UNIX or $tasks$ on NXT OSEK). This block can be ragarded as low-level control of execution process. It should be also marked that this block almost has no sence in interpretation mode (because execution itself is performed on desktop machine with only sending primitive commands to robot), but will be very useful in future works when autonomous mode will be introduced.

6. Future works

7. Conclusion

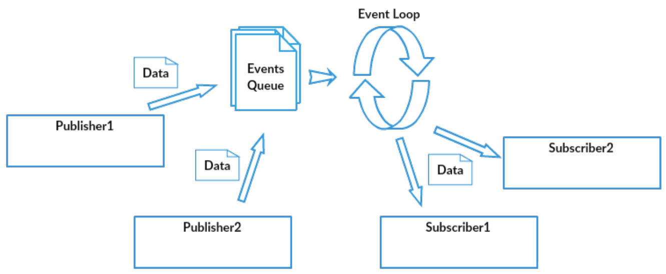

Figure 4: Proposed mechanism of pseudo-concurrent dataflow interpretation

References

[1] "IEEE International Conference on Robotics and Automation," 2016. [Online]. Available: http://www.icra2016.org/

[2] "International Conference on Intelligent Robots and Systems," 2016. [Online]. Available: http://www.iros2016.org/

[3] "IEEE Symposium on Visual Languages and Human-Centric Computing," 2016. [Online]. Available: https://sites.google.com/site/vlhcc2016/

[4] O. Banyasad, "A visual programming environment for autonomous robots," Master's thesis, DalTech, Dalhousie University, Halifax, Nova Scotia, 2000.

[5] J. Simpson, C. L. Jacobsen, and M. C. Jadud, "Mobile robot control," *Communicating Process Architectures*, p. 225, 2006.

[6] J. Simpson and C. L. Jacobsen, "Visual process-oriented programming for robotics." in *CPA*, 2008, pp. 365–380.

[7] J. C. Posso, A. T. Sampson, J. Simpson, and J. Timmis, "Process-oriented subsumption architectures in swarm robotic systems." in *CPA*, 2011, pp. 303–316.

[8] J. P. Diprose, B. A. MacDonald, and J. G. Hosking, "Ruru: A spatial and interactive visual programming language for novice robot programming," in *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*. IEEE, 2011, pp. 25–32.

[9] "NXT-G quick programming guide," 2013. [Online]. Available: http://www.legoengineering.com/nxt-g-quick-guide/

[10] "$AllaboutTRIK : TRIKstudio$," 2016. [Online]. Available: http://blog.trikset.com/p/trik-studio.html

[11] "ROBOLAB quick guide," 2013. [Online]. Available: http://www.legoengineering.com/robolab-quick-guide/

[12] W. M. Johnston, J. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM Computing Surveys (CSUR)*, vol. 36, no. 1, pp. 1–34, 2004.

[13] "LabVIEW System Design Software - National Instruments," 2016. [Online]. Available: http://www.ni.com/labview/

[14] "Simulink - Simulation and Model-Based Design," 2016. [Online]. Available: http://www.mathworks.com/products/simulink/

[15] "MINDSTORMS EV3 - Products," 2016. [Online]. Available: http://www.lego.com/en-us/mindstorms/products/

[16] J. Simpson and C. G. Ritson, "Toward process architectures for behavioural robotics." in *CPA*, 2009, pp. 375–386.

[17] M. Proetzsch, T. Luksch, and K. Berns, "The behaviour-based control architecture ib2c for complex robotic systems," in *KI 2007: Advances in Artificial Intelligence*. Springer, 2007, pp. 494–497.

[18] J. M. Gomez-de Gabriel, A. Mandow, J. Fernandez-Lozano, and A. J. Garcia-Cerezo, "Using lego nxt mobile robots with labview for undergraduate courses on mechatronics," *IEEE Trans. Educ.*, vol. 54, no. 1, pp. 41–47, 2011.

[19] J. Jackson, "Microsoft robotics studio: A technical introduction," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 82–87, 2007.

[20] S. H. Kim and J. W. Jeon, "Programming lego mindstorms nxt with visual programming," in *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*. IEEE, 2007, pp. 2468–2472.

[21] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1321–1326.