

Visual Dataflow Language for Educational 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 are inherently reactive: 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.

This paper discusses a novel extensible tool for programming all popular educational robotic kits on dataflow visual programming language. One of the most important distinction of our tool from others is its focus on embedded systems (sec. 6). Another interesting detail of our work is the application of DSM-approach for implementation of visual editor: it is entirely generated by QReal DSM-platform[16][17] without even a line of code written. We also take into consideration the popularity of Brooks’ Subsumption Architecture[18] which is still mainstream approach to design of complex robotic controllers[4], [5], [7], [19] despite it was proposed 30 years ago. Brooks Subsumption Architecture and some other are conveniently expressed in our language, they are discussed in sec. ??.

The remainder of a paper is organized as follows. An overview of robotics VPLs and DFVPLs is presented in section 2. Section ?? provides some general thoughts on how some widely used robotic behavioural architectures are expressed in our language. A detailed description of our language is given in section 4. Section 5 demonstrates two typical robotic controllers expressed in our language. The most important details of implementation are deiscussed in section 6. Finally, the last section concludes the paper and discusses possible directions for future work.

2. Similar Tools

Robot programming environments can be divided into three categories: educational, which allows to program small educational robotic kits; industrial, which have a rich toolkit for creating large and complex robotic controllers; academic, which implement new interesting ideas, however they are oftenly unavailable for downloading or unusable.

Educational visual environments are for example NXT-G and ROBOLAB for LEGO MINDSTORMS NXT kit, EV3 Software for the Lego Mindstorms EV3 kit, TRIK Studio for NXT, EV3 and TRIK. Those environments simplify solving primitive robot control tasks like finding a way out of the maze and driving along the line using light sensors, which makes the process of learning the basics of programming and robot control easy. But their simplicity often bounds the flexibility of the language. Visual languages of all mentioned systems are based on control flow model.

There is also a number of well-known visual robotic programming environments of industrial level. For example, general-purpose LabVIEW from National Instruments with the DFXPL G, programming environment Simulink developed by MathWorks for modeling different dynamic models or control systems. Those products offer to a huge set of models and libraries to create control systems, test benches, real-time systems of any complexity, using model-driven approach. LabVIEW provides opportunity for programming small robots. There are lots of examples of applying LabVIEW in education[20], [21], but much more often its adaptations like Robolab are used in educational process. It should be noted that those environments are distributed under the commercial license.

Another example of an visual robotics industrial system is the Microsoft Robotics Developer Studio (MSRDS)[22], which is free for academic purposes and allow to create distributed robotic systems on DFXPL. MSRDS officially supports a large set of robotic platforms, LEGO NXT[23] in particular (however, the autonomous mode for NXT is not supported). MSRDS has the ability of manual integration with custom robotic platforms, but unhappily is not maintained since 2014.

There is a lot of scientific research done in this area, e.g., dissertation[4] describes a visual programming module for expressing robotic controllers in terms of extended Moore machines, [6], [7] describe visual environment for *occam- π* language and *Transferpreter* framework, and its usage in education and swarm robotics. Article[8] describes DFXPL for beginners which is pretty close to a one we introduce here. However at the moment RuRu is under development, it has pretty limited functionality and even unavailable for download.

3. Robotic Behavioural Architectures

The task of creation complex and scalable robotic controller is indeed a non-trivial task. Starting from middle 80's many researchers have attempted to solve this problem and a number of behavioural robotic architectures were proposed[24]. Those approaches are quickly became popular in robotics community and they are still actual. For example the original work that introduced Brooks' *Subsumption Architecture*[18] is one of the most cited works in the entire robotics domain. We believe that the description of modern language for programming robotic controllers should contain at least general thoughts on how those architectures may be expressed in it.

A controller built on Brooks' Subsumption Architecture is decomposed into a hierarchy of levels of competence where each new layer describes a new feature of robot's behaviour. Levels are "ordered" upside-down, the higher levels describe more "intelligent" behaviour of robot. Higher levels depend on lower ones but not vice versa, so failures of higher levels do not imply the failure of lower. This is important feature for mobile robotics, e.g. if robot's gripper was damaged the controller is still able to deliver robot to its base. Levels of responsibility are expressed as a set "behaviours" running concurrently and interacting with each other via channels of *suppression* and *inhibition*. Using them higher levels can suppress the activity of lower ones thus correcting the behaviour of the whole system.

Brooks' in his original work offered to express behaviours in terms of *state machines*. Each layer implements some simple logic of transformation sensor inputs into impulses on actuators. Dataflow languages are obviously as suitable as state machines for expressing such behaviours. In our language each behaviour can be represented as "black box" described by separate subprogram. Also our language contains *Suppressor* and *Inhibitor* elements for layers communication. Levels can be invoked concurrently, so we can conclude that our language allows the convenient expression of controllers built with Subsumption Architecture. That is demonstrated by an example in section 5.

Connell's *Colony Architecture*[25] is a very similar to Brooks' one, but solves some scalability issues of Subsumption Architecture. It also decomposes the controller into a number of communicating concurrent levels, but they are unordered. The other difference is in an absence of inhibition channel, data inhibition should be implicitly expressed by predicated in layers. Our language does not force any order between layers, predicative inhibition can be implemented simply with *Filter* block. So Colony Architecture is also well-expressed in our language.

There also exist Arkins *Motor Schema*[26] and Rosenblatts *Distributed Architecture for Mobile Navigation (DAMN)*[27] which are compatible with our language, but the detailed descriptions will be omitted here. General ideas on their implementation on *occam- π* language can be found in [24], we believe that those ideas will suffice in the context of this paper. The complete research of expressing behavioural architectures in our language is a topic for separate paper.

4. Description of the Language

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

5. Example

6. 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 meta-model 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-approach is obvious, 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 infrastructure: Lego NXT or EV3 robot, TRIK robot, TRIK Studio 2D simulator or *V-REP* 3D simulator[28]. Commands are sent via high-level TRIK Studio devices API, a part of it presented at fig. 2.

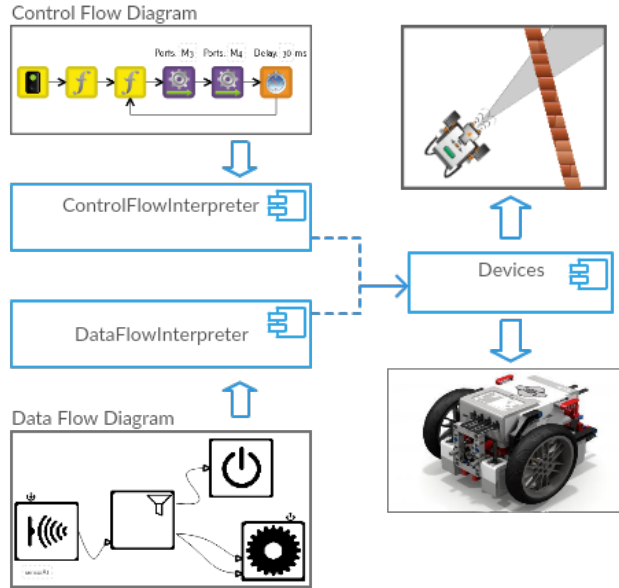


Figure 1: The general architecture of the system

The general architecture of interpreter 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 incoming flows. After all that done preparation phase is complete and diagram starts being

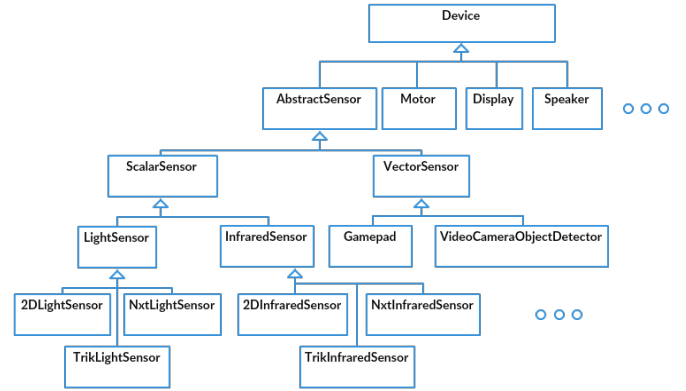


Figure 2: Partial architecture of devices used in dataflow interpreter

interpreted.

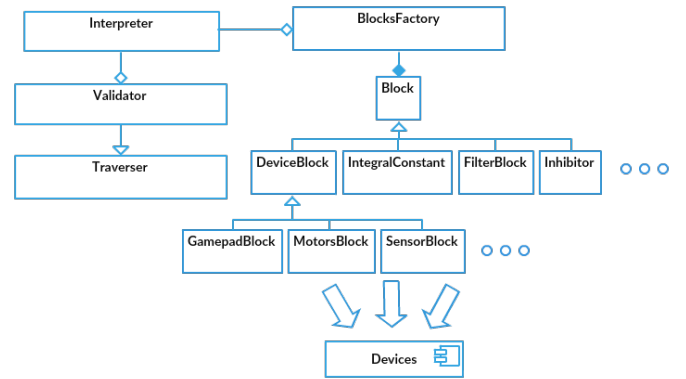


Figure 3: The general architecture of dataflow interpreter plugin

Interpretation process is not as straightforward as in most asynchronous dataflow environments. Usually components of dataflow diagram are executed concurrently, on different 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 our case we deal exactly with embedded devices (Lego NXT, EV3, TRIK, Arduino controllers), so we propose here another way of executing dataflow diagrams. The main idea is to introduce 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 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 flow processing. In our language that chain can activate control flow ports of blocks “reviving” them, so the control flow model is implicitly supported in our 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 heuristic 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 *pthread*s on UNIX or *tasks* on NXT OSEK). This block can be regarded as low-level control of execution process. It should be also marked that this block almost has no sense 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.

an approach for executing dataflow diagrams on embedded devices. The language implicitly supports control flow model for educational purposes. It is also convenient for expressing typical robotic controllers architectures which is demonstrated on example.

The implemented system can be regarded as a platform for future investigations. First of all autonomous mode of work will be implemented. That will be done through code generation into a number of textual languages already supported by TRIK Studio (NXT OSEK C for Lego, bytecode for EV3, JavaScript, F#[29] and Kotlin for TRIK). We are also interested in academical research. First of all a formal semantics of our language should be expressed for applying various formal methods of program analysis. Another branch of research will be directed into a DSM-branch, here we want to consider an ability of dynamic language meta-model generation from specifications of available modules of robotics middleware (like ROS[30] or Player[31]).

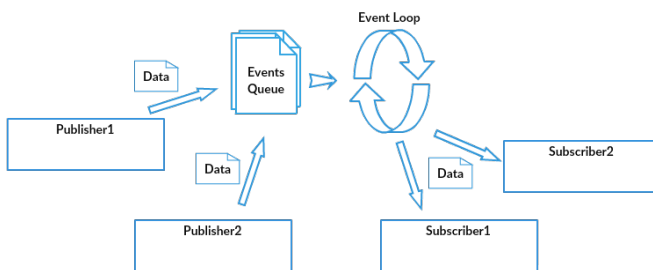


Figure 4: Proposed mechanism of pseudo-concurrent dataflow interpretation

7. Conclusion and Discussion

In this work we presented the prototype of dataflow language for programming different robotic kits (LEGO MINDSTORMS NXT, LEGO MINDSTORMS EV3, TRIK). The system provides ability to interpret diagrams on 2D- and 3D-simulators and real robotic devices. Proposed

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] "All about TRIK: TRIK Studio," 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] A. Kuzenkova, A. Deripaska, K. Taran, A. Podkopaev, Y. Litvinov, and T. Bryksin, "Sredstva bustroi razrabotki predmetno-orientirovannykh resheniy v metacase-sredstve qreal," *St. Petersburg State Polytechnical University Journal*, p. 142, 2011.
- [17] A. Kuzenkova, A. Deripaska, T. Bryksin, Y. Litvinov, and V. Polyakov, "Qreal dsm platform-an environment for creation of specific visual ides," in *ENASE 2013 — Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering*. Setubal, Portugal: SciTePress, 2013, pp. 205–211.
- [18] R. A. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.
- [19] 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.
- [20] B. Erwin, M. Cyr, and C. Rogers, "Lego engineer and robolab: Teaching engineering with labview from kindergarten to graduate school," *International Journal of Engineering Education*, vol. 16, no. 3, pp. 181–192, 2000.
- [21] 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.
- [22] J. Jackson, "Microsoft robotics studio: A technical introduction," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 82–87, 2007.
- [23] 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.
- [24] J. Simpson and C. G. Ritson, "Toward process architectures for behavioural robotics," in *CPA*, 2009, pp. 375–386.
- [25] J. H. Connell, "A colony architecture for an artificial creature," DTIC Document, Tech. Rep., 1989.
- [26] R. C. Arkin, "Motor schema based navigation for a mobile robot: An approach to programming by behavior," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4. IEEE, 1987, pp. 264–271.
- [27] J. K. Rosenblatt, "Damn: A distributed architecture for mobile navigation," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 339–360, 1997.
- [28] 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.
- [29] A. Kirsanov, I. Kirilenko, and K. Melentyev, "Robotics reactive programming with f#/mono," in *Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia*. ACM, 2014, p. 16.
- [30] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.
- [31] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.