Domain-Specific Approach to Software Development for Microcontrollers

Boris Sedov, Sergey Pakharev, Alexey Syschikov, Vera Ivanova
Saint Petersburg State University of Aerospace Instrumentation
Saint-Petersburg, Russia
{boris.sedov, sergey.paharev, alexey.syschikov, vera.ivanova}@guap.ru

Abstract—**Microcontrollers are widely used in many areas of embedded systems, from robotics control systems to smart homes. The number of different hardware platforms is increasing with a spread of embedded systems. More and more users are involved in a software development, including not qualified programmers. The application of a domain-specific development technology provides the possibility to program microcontrollers in terms that are familiar to a user but not to a microcontroller. The visual representation of programs ensures the clarity of the processing workflow. A portability to different hardware platforms allows using the microcontrollers with different processors from different manufacturers and an easy switching to the new versions of microcontrollers. In this paper we present initial steps of working with VIPE toolset for support microcontrollers programming for embedded systems.**

I. INTRODUCTION

Nowadays microcontrollers have a tendency to enter the various scopes of activity. Embedded systems, used in microcontrollers, unite in computing network conceptions, such as internet of things, internet of energy or internet of vehicle. In the short term, it is planned to unite all of this interconnected autonomous environments into common cyber-physical systems.

Embedded systems perform monitoring and control of physical components and processes in different fields. Therefore, more people with different levels of programming skills will interconnect with them. There is a need in tools and technologies that will allow creating applications for such systems to more and more people with different skills background.

Today there is a tendency to use a visual approach to a microcontrollers programming. The visual approach has become very popular due to its low cognitive resistance [1], which lowers greatly the required minimum level of user special skills of a microcontrollers programming. Visual languages can be adapted to different user needs with taking into account the level of their skills, features of applications and application domains.

The best-known tool that implements the visual approach and is designed especifically for a microcontrollers programming is Lego Mindstorms EV3 programming software [2] (Fig.1). Modkit [3] (Fig.2), TRIK Studio [4] (Fig.3) and some other tools are also known quite well. However, a small number of supported hardware platforms restricts all of those tools. They are purposed mainly for learning and education in robotics. It affects the aspects of theirs visual approach. The figures below illustrates that the main accent of these environments is set to colorfulness, scheme simplifying and effectiveness of small programs representation. When programs come to significant functionality and large size, the effectiveness of an implemented approach is lowering drastically.

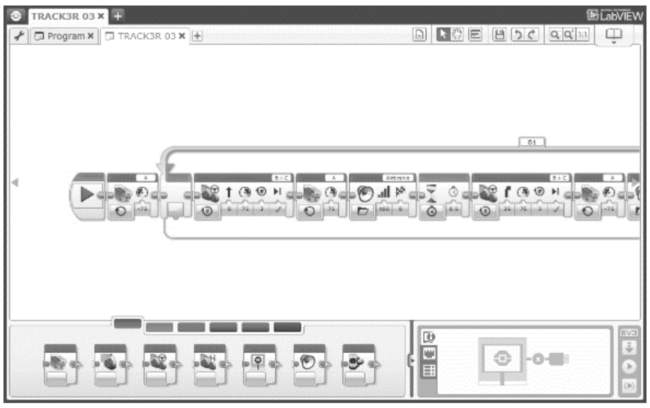
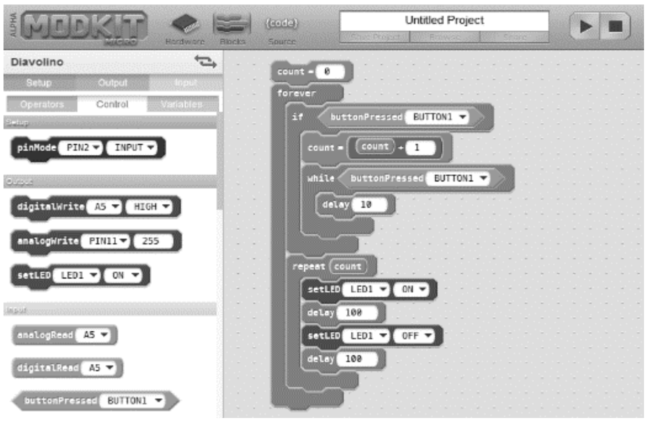


Fig.1. Mindstorms environment



```
count = 0
forever
  if buttonPressed BUTTON1
    count = count + 1
    while buttonPressed BUTTON1
      delay 10
  repeat count
    setLED LED1 ON
    delay 100
    setLED LED1 OFF
    delay 100
```

Fig.2. Modkit environment

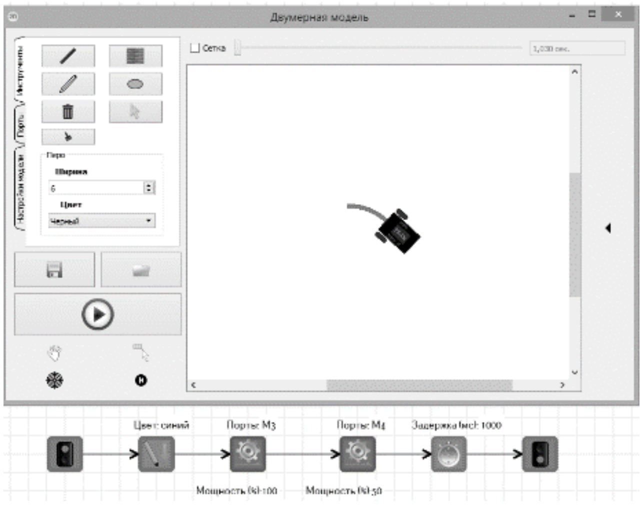


Fig.3. TRIK Studio environment

The visual approach is supported by major players of a software market, such as Mathworks (Simulink, [5]), Microsoft (Microsoft Robotics Developer Studio, MRDS, [6]) and National Instruments (LabVIEW, [7]). All of these systems are essentially similar, and have similar key drawbacks from the point of embedded microcontrollers programming. Firstly, it is limited support of microcontroller platforms and closed nature of these systems: only system owners can develop the support of new microcontrollers. Secondly, lack of convenient ways for creation of domain-specific languages or libraries. For example, MRDS is suited for robots programming, which lowers its effectiveness in a programming of tasks that differs from robotic control systems. Simulink is more suited for dynamic systems design. LabVIEW is generally suited for creating virtual instruments and so on. As a result, when working with other domains development, an efficiency decreases.

The visual interactive developing environment VIPE is intended for an algorithms design and programming in various data processing domains. VIPE technologies are based on coarse-grained visual approach, which allows separating design and programming processes. A task developer designs the structure of blocks and determines their interaction with each other without thinking on a particular implementation of each block. The programmer write the code of each block directly without thinking on a structure of the entire software complex. Detailed description of this approach is presented in [8]. In addition, VIPE provides support for the development of visual domain-specific languages [9].

During the VIPE development, the main exploitation was performed in data processing domains. To review the scope and breadth of an application, this paper presents the process of applying the technology to a microcontrollers programming domain. The process includes the creation of DSL for one of the most popular platform for non-industrial microcontrollers – Arduino, ensuring the execution of program schemes on the hardware platform, and use cases of programs development with the designed tools.

At the end of the paper we will review the perspectives and methods of DSL and tools universalization to support larger number of controllers from other manufacturers, and ensuring the portability of developed program schemes.

II. DEVELOPMENT OF DOMAIN-SPECIFIC LANGUAGE FOR ARDUINO PLATFORM

A. Domain analysis

The first stage of work with a new domain – the design of domain-specific language. “Piggyback” method is used for DSL design in the considered technology. According to this method, host language constructions are enhanced with new domain-specific constructions. This and other methods of a language design are discussed in [9].

We performed the domain analysis. Arduino platform is based on AVR ATmega microcontrollers by Atmel [10, 11]. In addition to the microcontroller itself, the platform includes basic peripherals, such as digital and analog I/O ports. The analysis has shown that in addition to the basic environment for programming ATmega microcontrollers, extended software toolset has been created for the Arduino platform. So, the specifics of an Arduino platform programming is already reflected in the basic Arduino text programming language (C/C++ based).

The basis of a DSL design is:

- basic constructions of a text language;
- functions for working with embedded peripherals;
- functions for working with the popular external peripherals (actuators, proximity sensors, light sensors, accelerometers, etc.).

B. Constants

The Arduino software toolset includes the number of named constants. They are not obligatory for DSL, but they are familiar to Arduino developers. The main constants, for example, HIGH and LOW, that specifies values for digital I/O, are put into the VIPE constants table. This table is available from any place in a program.

C. Language elements

Base control structures (conditions, cycles and so on) are already presented in the host VPL language. Arithmetic and logic operations are contained in the base VPL libraries and do not require an additional inclusion in the developing DSL.

Arduino functions are grouped around entities with which they are interacting.

The main function group is working directly with Arduino board. This group includes the function pinMode, which is responsible for setting the port as input or output. A language element was established for this function (Fig.4). In addition, language elements were created for other typical operations directly on the board: setting a port voltage level, reading a voltage value from a microcontrollers' pin, working with a timer, delay functions, working with COM-port etc.

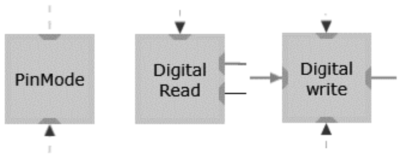


Fig.4. pinMode, digitalRead, digitalWrite functions

To provide the operation of plug-in peripheral modules it was necessary to provide elements to work with their libraries. Actuators, ultrasonic sensors, accelerometers and some other devices were chosen as primary devices.

Language elements for servo library were created to allow working with actuators. Elements of sensor initialization and range estimation were created for working with ultrasonic sensor. Elements for working with a display, functions of getting and processing values from accelerometers and gyroscopes were also created.

As a result, 28 language elements were specified, including the elements to initialize contacts, a servo control and other modules. Together, they forms a new DSL for Arduino (Fig.5).

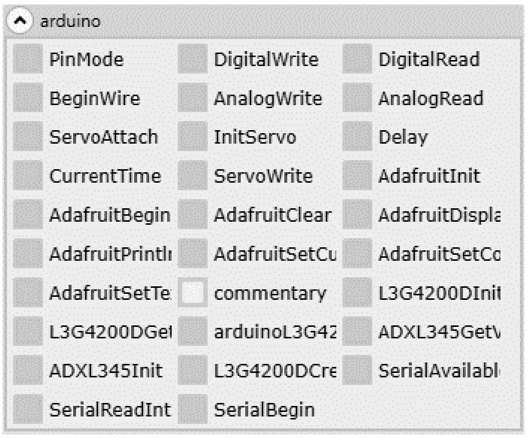


Fig.5. Arduino visual DSL library

Definitely, the developed DSL is not full, because the Arduino platform has lot of external devices developed for it. Moreover, the DSL does not include any data processing functions. The considered technology is open for DSL design and developed language can be easily extended in future by any person.

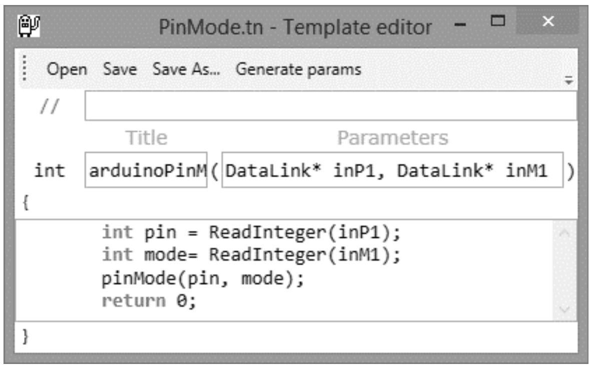
D. Target platform support

The technology of code generation allows working with a target platform. This technology is presented in details in [8]. It is important to notice that a target platform support is not necessary directly for programming with the DSL. It is possible to start working on an application software and develop a target platform support concurrently, lowering the time to market for new products.

Since the Arduino platform toolkit uses C/C ++ with some extensions, the development of a new code generator is not required. We can use the basic C/C ++ code generator, supplied with the toolset of considered technology.

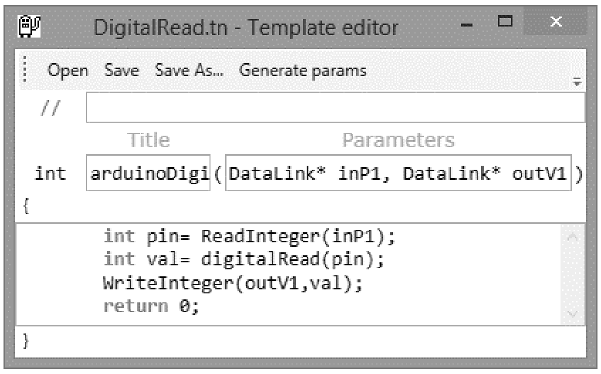
To ensure the functioning of a new DSL, it is necessary to develop function templates that implement elements of the language. The template mechanism that is implemented in the proposed technology allows attaching the textual function implementations, written in the target platform language (in our case - C/C++) to the graphical DSL elements.

The main group of platform functions has direct prototypes in Arduino software toolset functions. For these functions, the templates will be just wrappers on existing functions. Here are the examples of templates for pinMode (Fig.6), digitalRead (Fig.7) and millis (Fig.8).



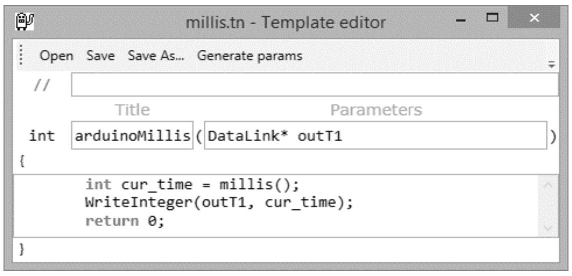
```
//
int arduinoPinM( DataLink* inP1, DataLink* inM1 )
{
        int pin = ReadInteger(inP1);
        int mode= ReadInteger(inM1);
        pinMode(pin, mode);
        return 0;
}
```

Fig.6. Template for pinMode function



```
//
int arduinoDigi( DataLink* inP1, DataLink* outV1 )
{
        int pin= ReadInteger(inP1);
        int val= digitalRead(pin);
        WriteInteger(outV1,val);
        return 0;
}
```

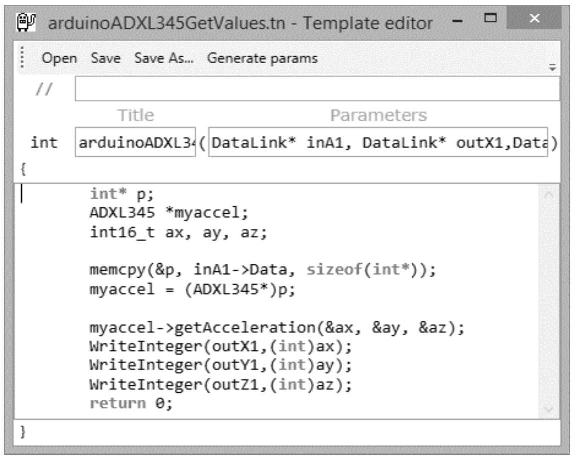
Fig.7. Template for digitalRead function



```
//
int arduinoMillis( DataLink* outT1 )
{
        int cur_time = millis();
        WriteInteger(outT1, cur_time);
        return 0;
}
```

Fig.8. Template for millis function

To provide operations with plug-in modules it is necessary to develop an interaction with their libraries and data processing functions. For example, data obtaining function arduinoADXL345GetValues (Fig.9) and data processing function arduinoKallmanTranslateAngle (Fig.10) were created to work with accelerometers and gyroscopes. First one gets data from module and the second one performs raw values transformation to obtain angle values.

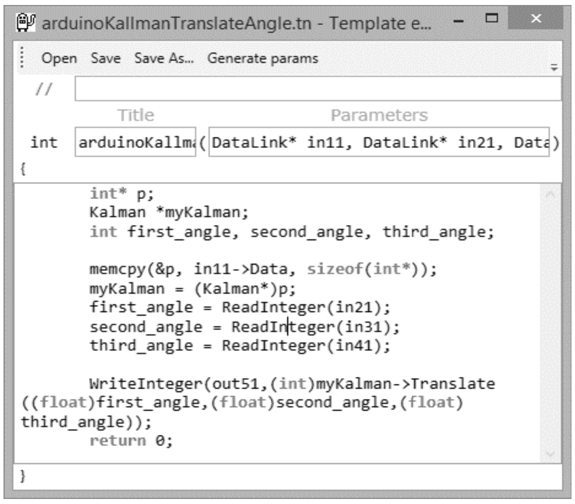


```
//
int arduinoADXL34( DataLink* inA1, DataLink* outX1,Data)
{
        int* p;
        ADXL345 *myaccel;
        int16_t ax, ay, az;

        memcpy(&p, inA1->Data, sizeof(int*));
        myaccel = (ADXL345*)p;

        myaccel->getAcceleration(&ax, &ay, &az);
        WriteInteger(outX1,(int)ax);
        WriteInteger(outY1,(int)ay);
        WriteInteger(outZ1,(int)az);
        return 0;
}
```

Fig.9. Template for arduinoADXL345GetValues



```
//
int arduinoKallm( DataLink* in11, DataLink* in21, Data)
{
        int* p;
        Kalman *myKalman;
        int first_angle, second_angle, third_angle;

        memcpy(&p, in11->Data, sizeof(int*));
        myKalman = (Kalman*)p;
        first_angle = ReadInteger(in21);
        second_angle = ReadInteger(in31);
        third_angle = ReadInteger(in41);

        WriteInteger(out51,(int)myKalman->Translate
((float)first_angle,(float)second_angle,(float)
third_angle));
        return 0;
}
```

Fig.10. Template for arduinoKallmanTranslateAngle

Templates for all 28 elements of the DSL have been implemented in the similar way.

E. Program structure

The analysis of a program structure requires a special attention. Unlike the classic C/C++, a text program for Arduino contains two main functions: setup() and loop(). The first one, setup(), is performed only once. It is intended for specifying the initial configuration of a board, I/O, connected modules and so on. The loop() function is executed iteratively and contains the main program, which implements the core functionality.

The constructs of the host VPL language are enough to represent an overall structure of the program (Fig.11). The structural operator Complex will be used for the setup() structure. It ensures the single execution and can include other operators. The loop operator While will be used for the loop() structure, providing an endless iterative execution. The While operator also can include other operators.

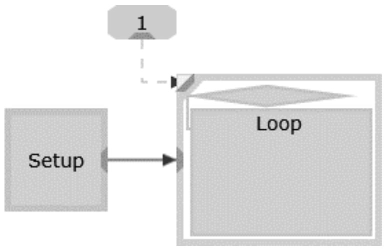


Fig.11. Program structure in VIPE

III. APPLICATION OF ARDUINO DSL

As a use-case for developed DSL for Arduino the task of an automated control of a radio-controlled vehicle model has been selected.

Arduino platform with the ultrasonic sensor was installed on COTS radio-controlled model (Fig.12).

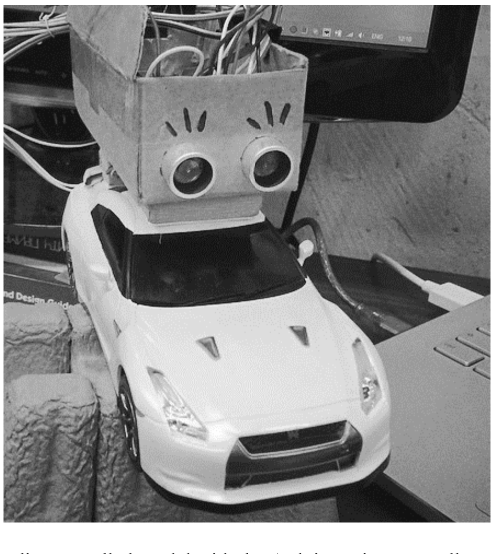


Fig.12. Radio-controlled model with the Arduino microcontroller

The model is controlled manually from the control panel. Arduino monitors the area in front of the vehicle with an ultrasonic sensor. In the case of detecting an obstacle, Arduino blocks the control of acceleration and enforces the emergency brake to avoid a collision.

A. Visual program structure

The common view of the control system program is shown on the Fig.13.

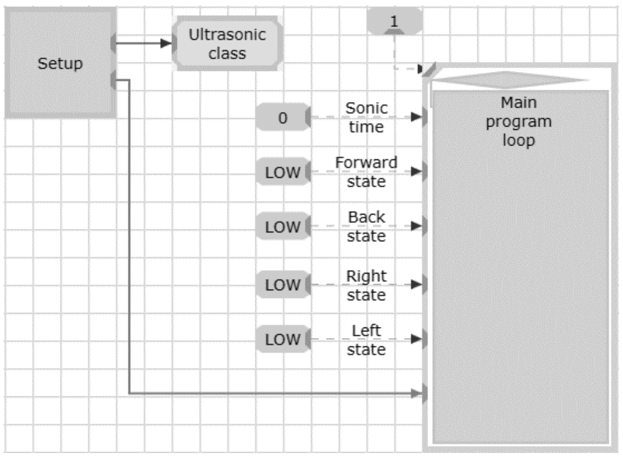


Fig.13. A general view of control system scheme

All necessary functions to configure a platform (Fig.14) is placed in the Setup block: ports configuration, proximity sensor initializing and so on.

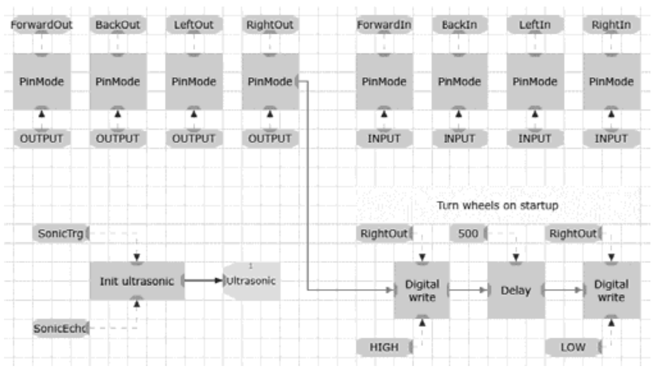


Fig.14. Setup block structure

The main part of the control system (Fig.15) is placed in the infinite loop block "Main program loop".

It can be easily seen on the Fig.15 that base constructions of VPL language, such as structural operators, conditional operators, constants and virtual nodes – are used for structuring the program and for miscellaneous operations.

In addition, elements of the host language and base libraries are used to organize common computations and data processing (Fig.16).

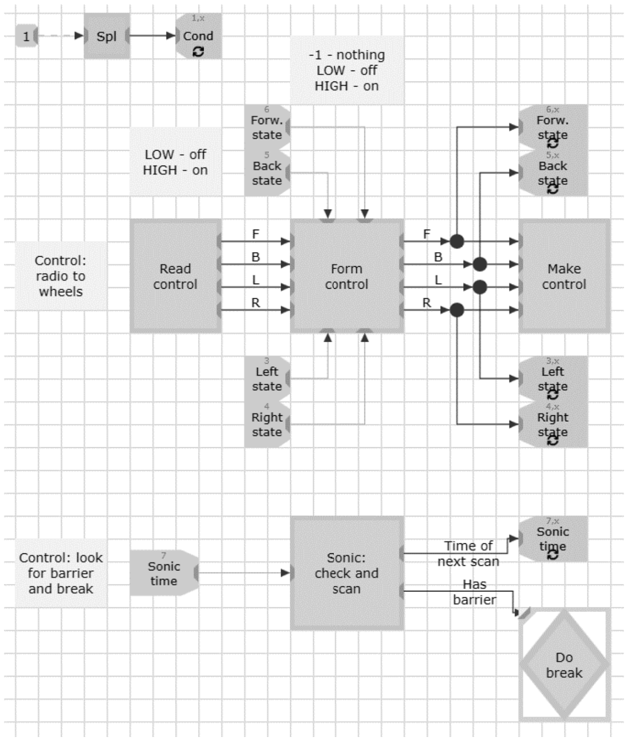


Fig.15. The main scheme for vehicle model control

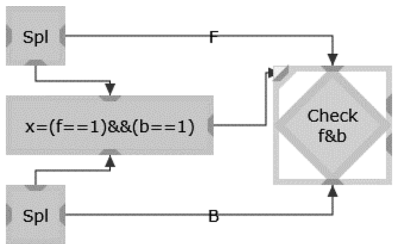


Fig.16. The scheme with computations

The interaction with domain-specific aspects is performed by using the DSL elements (Fig.17).

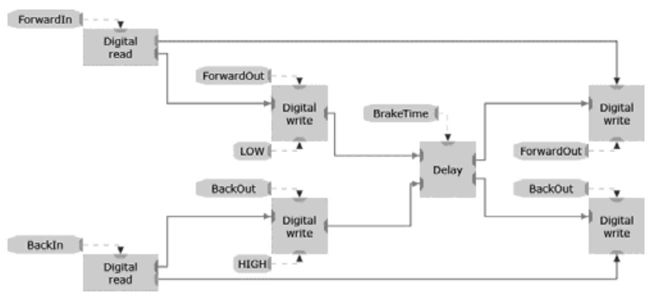


Fig.17. The scheme with DSL blocks

B. Deployment to target platform

A target platform code was generated for the developed scheme using templates of base VPL libraries and Arduino DSL. The generated code is fully compatible with the toolset provided with the Arduino platform. The code has been successfully compiled, uploaded to the platform and correctly executed.

Compiling of a binary file and its loading to the platform was integrated directly into VIPE development environment for the convenience of developers (Fig.18). Thus, the developers are fully freed from the necessity to interact with several tools, such as a compiler, loader, Arduino development environment etc. They work within the single visual environment VIPE.

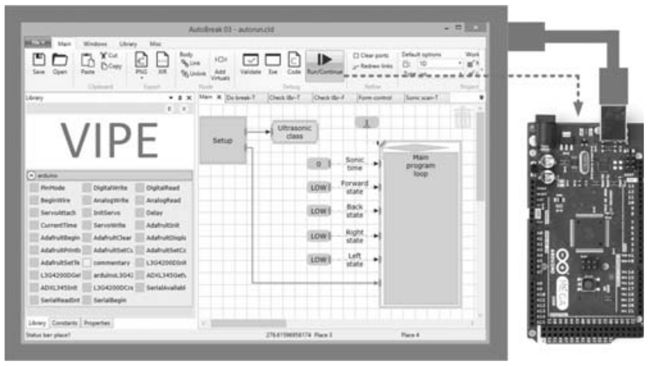


Fig.18. Integration of Arduino tools calling in VIPE

It should be noted that the size of a generated code is quite large. It is 45Kb against 4Kb of the manually written code. It is a consequence of two aspects. Firstly, the code generator and templates structure are tailored for coarse-grained approach. For medium-grained and fine-grained Arduino schemes, a service code is an essential part of the final text of the program. Secondly, the schemes and generated code are focused on a parallel execution (Fig.19), which is not required for single core microcontrollers.

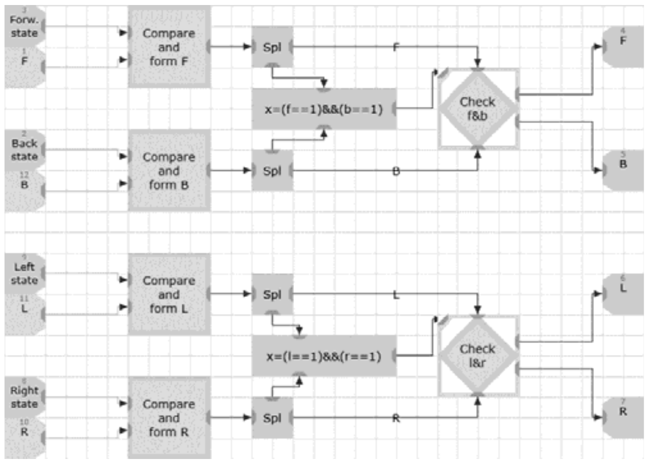


Fig.19. Parallel program scheme

The solutions for a generated code size optimization are under development and will be introduced in the nearest future.

IV. RESULTS AND FUTHER PLANS

The DSL for Arduino platform has been developed within the technology of DSL creation [8] and with the instruments of integrated environment VIPE. The control system was successfully ported to the target hardware platform using the ready tools of the considered technology.

The development experience has shown that the considered approach and technology supporting tools provide fast, effective and low-cost way to design new DSLs for new application domains. Technology mechanisms allow extending existing DSL with new elements if it is required. The usage of the created DSL allows designing programs within this domain with a use of the visual medium-grained approach. The integrated software complex provides the uniform environment, freeing from the use of several separate tools. In this way it was achieved the ability to program microcontrollers in terms familiar to the user, making the possibility of creating schemes for Arduino even easier. It significantly expands the opportunity to realize ideas on the Arduino board for people who are not skilled microcontroller programmers.

An important feature of the proposed approach is the possibility of developing a portable software. Adaptation methods for code generators and templates should provide a portability of developed schemes to multiple target hardware platforms.

We have analyzed other microcontroller hardware platforms that are close to the Arduino platform. The main players in this field are companies like FreeScale and STM (FRDM-KL25Z [12] and STM32F10X [13] platforms based on ARM Cortex M0+ processors) and Intel (Galileo platform [14] based on Intel Quark SoC X1000 processor).

The exploration showed that the developed DSL could not be effectively applied for porting programs to other microcontroller platforms. It was found that during the development of the DSL a significant drawback was made at the domain analysis phase. The domain is the programming of control systems based on microcontrollers. During the development of the DSL, this area was erroneously narrowed to the task of programming Arduino platform. As a result, developed DSL reflects the Arduino platform aspects too detailed.

In the nearest future, the developed DSL will be significantly revised. The identified shortcomings will be addressed and the work on the portability of program schemes will be continued.

Flexibility of the VIPE and the domain-specific programming technology allows using them not just for microcontrollers, but also for various application areas: smart systems, telecommunications, and so on.

ACKNOWLEDGMENT

The research leading to these results has received funding from the Ministry of Education and Science of Russian Federation under agreement n°14.575.21.0021, identifier RFMEFI57514X0021.

REFERENCES

[1] Joachim Booth, Tracey, and Simone Stumpf, *End-user experiences of visual and textual programming environments for Arduino. End-User Development*. Springer Berlin Heidelberg.

[2] Rollins, Mark, *Beginning Lego Mindstorms Ev3*. Apress, 2014.

[3] Millner, Amon, and Edward Baafi, "Modkit: blending and extending approachable platforms for creating computer programs and interactive objects", *Proceedings of the 10th International Conference on Interaction Design and Children. ACM*, 2011.

[4] Terehov A.N., Litvinov Y.V., Briskin T.A, "Environment for teaching informatics and robotics QReal:Robots", *9-th independent conference «Software deveopment 2013» (CEE SEC(R)-2013),* 2013.

[5] Gartseev, Ilya B., Leng-Feng Lee, and Venkat N. Krovi, "A low-cost real-time mobile robot platform (ArEduBot) to support project-based learning in robotics & mechatronics", *Proceedings of 2nd International Conference on Robotics in Education (RiE 2011), R. Stelzer and K. Jafarmadar, Eds. INNOC Austrian Society for Innovative Computer Sciences,* 2011.

[6] Johns, Kyle, and Trevor Taylor, *Professional microsoft robotics developer studio*. John Wiley & Sons, 2009.

[7] Travis, Jeffrey, and Jim Kring, *LabVIEW for Everyone: Graphical Programming Made Easy and Fun (National Instruments Virtual Instrumentation Series)*. Prentice Hall PTR, 2006.

[8] Boris Sedov, Alexey Syschikov, Vera Ivanova, "Technology and Design Tools for Portable Software Development for Embedded Systems". *Proceedings of the 16th Conference of Open Innovations Association FRUCT printed by "University Telecommunications" Company*, 2014, pp. 86-93.

[9] Vera Ivanova, Boris Sedov, Yuriy Sheynin, Alexey Syschikov, "Domain-Specific Languages for Embedded Systems Portable Software Development". *Proceedings of the 16th Conference of Open Innovations Association FRUCT printed by "University Telecommunications" Company*, 2014, pp. 24-30.

[10] Atmel official website, Web: http://www.atmel.com/

[11] Barnett, Richard, Sarah Cox, and Larry O'Cull, *Embedded C programming and the Atmel AVR*. Cengage Learning, 2006.

[12] Tadi, Mojtaba Jafari, et al, "Accelerometer-Based method for extracting respiratory and cardiac gating information for dual gating during nuclear medicine imaging", *Journal of Biomedical Imaging*, 2014.

[13] KU, Shao-ping, and Jing LIU, "Design of the Control Systems of Stepping Motor Based on STM32F10x and MDK [J]", *Journal of Wuhan University of Technology 3*, 2009.

[14] Ramon, Manoel Carlos, "Intel Galileo and Intel Galileo Gen 2", *Intel® Galileo and Intel® Galileo Gen 2 Apress*, 2014, pp 1-33.