Technology and Design Tools for Portable Software Development for Embedded Systems

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

***Abstract*—Nowadays embedded systems are used in broad range of domains such as avionics, space industry, automotive, mobile devices, domestic appliances and so on. There is enormous number of tasks that should be solved using embedded systems. There are many tools and approaches that allow developing of software for domain area experts, but mainly for general purpose computing systems. In this article we propose the complex technology and tools that allows involving domain experts in software development for embedded systems. The proposed technology has various aspects and abilities that can be used to build verifiable and portable software for a wide range of embedded platforms.**

I. INTRODUCTION

Why do we need such technology?

The performance of embedding systems is increasing and they are used in more and more domain areas. The complexity of tasks running on such systems is increasing rapidly. There is a transition from simple control systems to high-performance systems for image, audio, video processing [1,2] and even decision making [3,4]. An algorithmic and a programming complexity of such tasks also increase fast. Embedding systems programming specialists are no longer enough for solving such complex tasks. A "two-in-one" developer is required: skilled domain experts, who can solve a task, and a skilled programmer, who can implement the algorithm efficiently. For specific embedded platforms such developer additionally should have a deep understanding of platform development features. Our technology allows involving domain experts into development process even without advanced skills in programming.

Requirements to hardware platforms are always contradictive. From a customer's point of view the platform should be small, cheap, with low power consumption, and it should work perfectly. From a developer's point of view hardware platform should be fast, have a lot of memory and other resources. In fact, unlike general purpose systems, embedded system must solve allocated tasks according to requirements and constraints. The performance and resources should be sufficient for this and system should have minimum of unnecessary resources. Our technology allows performing an estimation and evaluation of program requirements and hardware platforms, providing information for selection of platform type and agreement of platform characteristics.

The development of an algorithm and program should be started before the selection of a specific platform configuration is make to provide small time to market for a completed hardware-software solution. During the development process and exploitation the requirements can be (and will be) changed. It may lead to changes in hardware platform and its characteristics. Also the single task can have dramatically different characteristics and platform requirements depending on aspects of its application. So it is possible that some program will be applied on several platforms with different configurations and different requirements to performance and other characteristics. Our technology can provide various task allocations to platforms depending on available resources and computation requirements. It also provides estimations of final hardware-software solutions characteristics.

Hardware platforms become out-of-date rapidly, but computation tasks that are executed on them changes rarely. Permanent change of hardware platform generations and new configuration releases leads to permanent porting of embedded software to new platforms. It also forces to support a number of platforms of different generations with different characteristics. Our technology provides solutions portability to a wide class of platforms and their configurations, and also technologies for porting solutions onto other platforms.

II. TECHNOLOGY OVERVIEW

A. Technology structure

The technology of portable embedded software development for a heterogeneous manycore embedded systems provides a full cycle of software development for embedded systems – from initial design stage to a ready to use code for a particular hardware platform.

ISSN 2305-7254

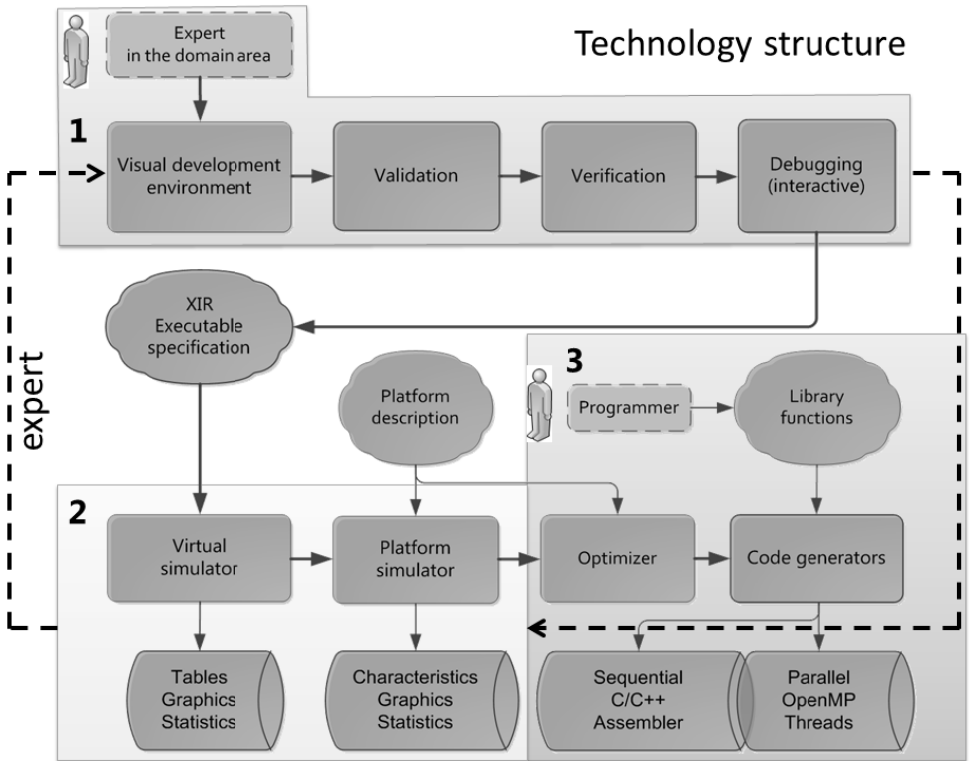

Fig. 1. Technology structure

There are some trends on market [5, 6, 7], which are similar on some way for proposed technology. One focused on system-level modeling, not on programming. Other try to mix design methodology with formal models (try create formal language). But no complex solution is detected.

The presented technology consists of three basic stages (Fig. 1):

- Algorithms design and programming.
- Simulation and early estimation.
- Deployment to target platforms.

Each of these stages is automated and has supporting tools. The process is organized in a way that a developer has an access to the results of each stage. It makes the development process completely controllable.

Every component of the technology will be presented in details below.

B. A program lifecycle

A lifecycle of the program development consist of three key stages (Fig. 2).

- Program development and debugging.
- Preparation for a deployment to the selected platform(s).
- Deployment to the various platforms.

At the first stage a domain expert design an algorithm, develop and debugged a functional program. Design is performed in the visual development environment using the platform-independent execution tools.

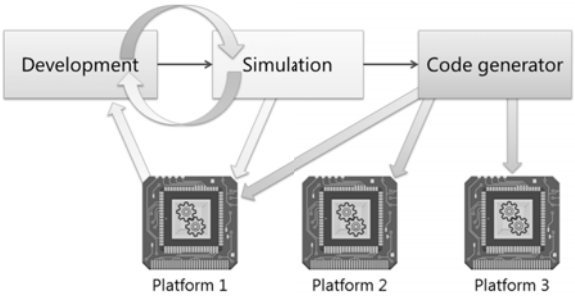

Fig. 2. A lifecycle of the program development

In fact, this stage corresponds to general-purpose programs development using traditional development technologies. The goal of this stage is getting a fully functional, algorithmically correct and debugged program. At this stage the simulator can be used to get platform-independent metrics of a program. An iterative process can be also applied for characteristics refinement. Engineers have been using models in systems development processes for years. What is changing as more companies adopt model-driven development flows is the ability to take these

models and use them more efficiently and effectively throughout the design process [8].

The second stage starts when the information about the hardware platform architecture, structure and characteristics becomes available for a developer. At this stage debugged program is executed on the platform simulator. The simulator provides a collection of platform-depended metrics based on a modelling of a program execution on a high-level platform model.

The developer can iteratively refactor the program using metric analysis results until required performance characteristics will be achieved. The simulator allows using different platform configurations that makes possible preparing the program for execution on all of chosen hardware platforms which configurations are known.The last stage is a deployment of the developed program directly to the chosen hardware platforms. Within a technology this deployment is performed by code generators. The code generator produces a code for the selected program scheme; this code will be used as an input for a particular hardware platform compiler.

To port the program to the new target platform a code generator should be developed. However, the complexity of such development is reasonable, well-predicted and allows making precise estimation of financial and time costs. Once developed the code generator could be actively used, so the costs become much less than a program development for each platform configuration separately.

III. Visual development environment

Visual development environment (VIPE) is designed for development of the computational tasks. VIPE provides the technology and instrumental support for algorithms design, programming and debugging.

A. Visual programming language

Domain expert develops solutions for the task that would be run on an embedded system. He uses the visual environment for algorithms design (Fig. 3).

VIPE is used for algorithms design and tasks programming in various domains areas. The basic instrument which is used for program development in VIPE is the Visual Programming Language (VPL) [9].

VPL is an “action language” [8] that can be both executed as a part of the model and translated into other languages. VPL language is based on Asynchronous Growing Processes (AGP) [10] formal computational model. AGP model defines the language syntax, semantic of language objects and construction of control units. The

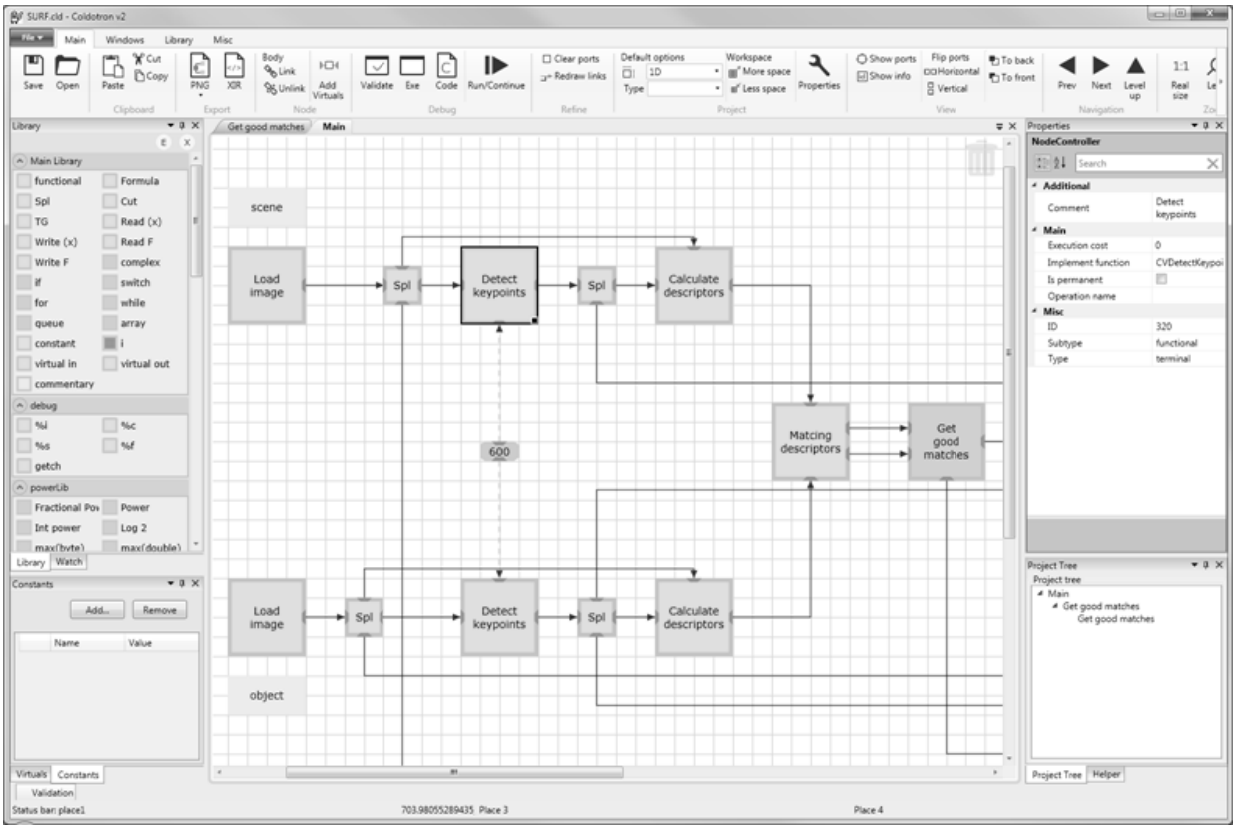

Fig. 3. General view of the visual development

formal model provides the set of important properties of the technology, such as: formal verification, debugging and portability. The influence of the formal model to these characteristics will be described in the next sections.

The investigation of the visual approach had shown its great potential. It has cognitive advantages for developer, such as clear view of development process, traceability of the dependency graph and the calculations management structure, natural parallelism and potential pipelining [11].

Visual approach allows a domain expert to concentrates on a task solving by operating on visual images of the domain area. He has no need to be a professional programmer. This fact allows involving more domain experts in the embedded systems software development.

Coarse-grained visual approach of the development process gives the opportunity to separate a design process from a programming (Fig. 4).

The domain expert designs a set of program blocks and defines their relationships. There is no need for a domain expert to understand implementation details of each block. A programmer writes code for each block independently, without full knowledge of the whole program structure.

Moreover the coarse-grained visual approach and a development separation provides following advantages:

- Easy program development.
- Easy changes in program structure.
- No direct programmer influence on a program scheme.
- Local appearance of code errors.
- Possibility of auto-documenting of the program scheme.
- Effective program maintenance during the whole lifecycle.
- Decreasing of errors possibility without sacrificing program obviousness.
- Flexibility and ease-of-change on any design stage.

B. Domain specific programming

The visual environment allows domain specific languages (DSL) designing for a particular domain based on coarse-grained libraries [12]. DSL libraries provide convenience of design within an application domain (Fig. 5) and enable re-use of development results. The more coarse-grained libraries of different domains are developed, the wider possibilities of results application appear.

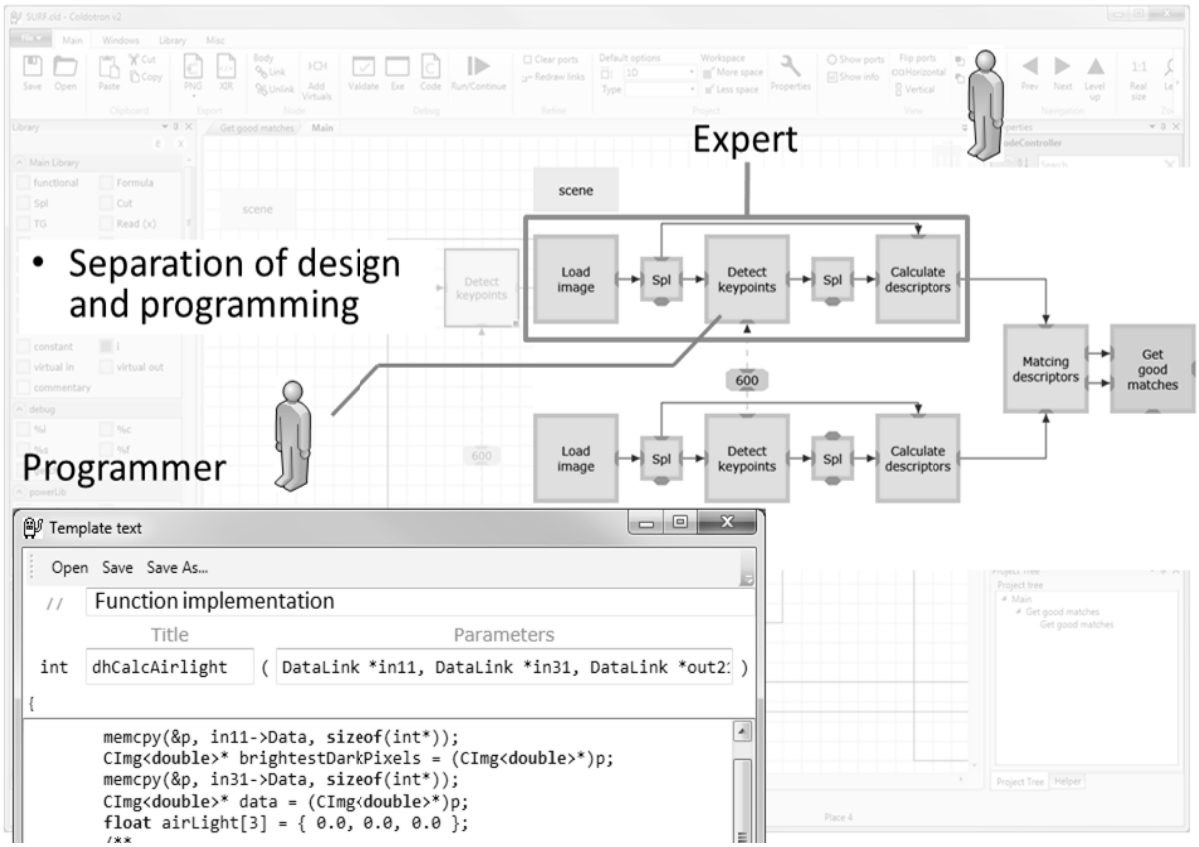

Fig. 4. Separation of design and program

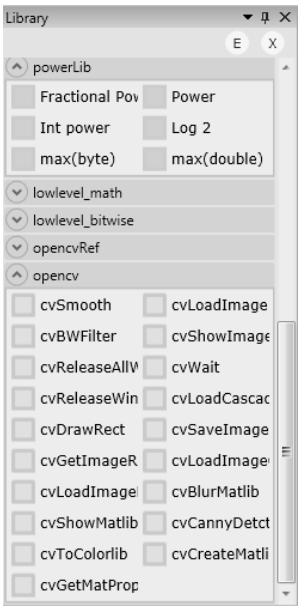

Fig. 5. Coarse-grained library for image processing

C. Interactive tools

The development of computational tasks cannot be performed without traditional development process support tools. VIPE is not an exception for this: it provides to developers the validation and interactive debugging tools. Proposed tools apply features of the implementing approach, collecting and displaying the most significant data for a developer.

Validation ensures syntactic correctness of developed visual programs. (Fig. 6).

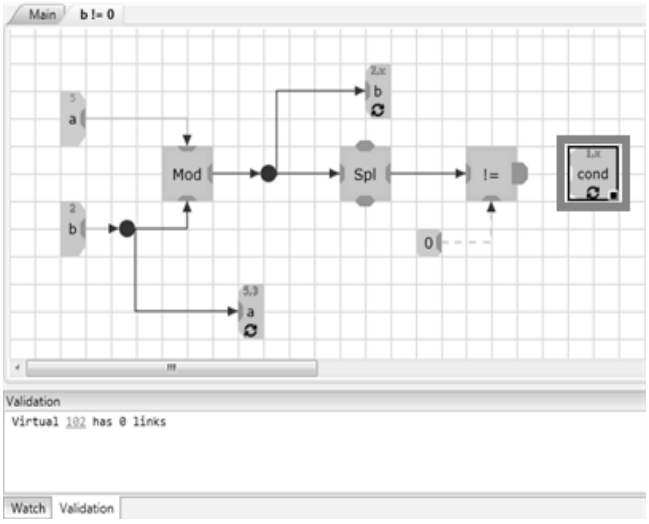

Fig. 6. Program scheme validation

The debugging tool provides well-known interactive functions, such as step-by-step debugging, breakpoints, watches etc. Besides this tool supports the features of an applied approach and enables an access to such information as history of data passing though the links, history of object execution, paths, traces etc. (Fig. 7).

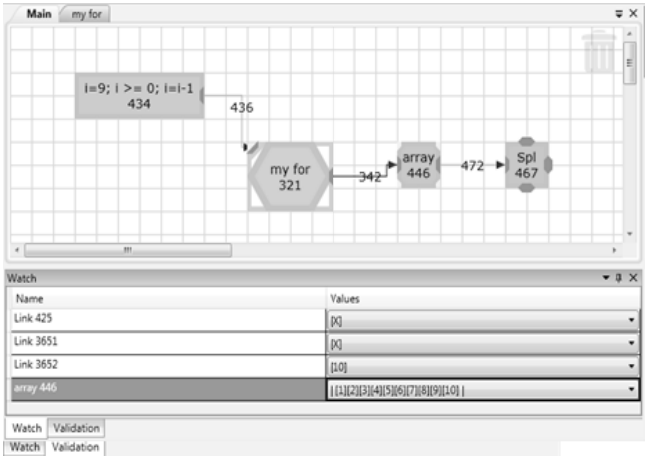

Fig. 7. Interactive debugging of the program

Implementation of the formal model-based approach allows providing the developer with a formal verification tool which verifies formal programs correctness “by design”.

The visual development environment allows designing a fully functional, debugged and algorithmically correct program. The developed program is platform-independent.

The major feature of this technology is deep formal model integration. Used formal AGP model allows spread the development and debugging program results on any form of a program execution. Formal model ensures that program will run similarly and will get similar results with no matter what kind of runtime environment (parallel, sequential, shared memory, distributed memory etc.) will be used.

IV. Early estimation and evaluation

The tools for early estimation and evaluation are critically important for reducing project schedules and improving a result quality [13]. They are used to analyse the particular task solution characteristics. They are oriented on an interactive process of characteristics analysis and a program scheme refinement (Fig. 8).

Main instruments of early estimation and evaluation are the virtual simulator and the platform simulator.

A. Virtual simulator

A virtual simulator bridges the gap between the hardware and software domains. It is both an abstraction of the design as well as a simulation environment. A platform simulator provides a representation of the design that can be used at early stages of detailed development work to question, validate, and test.

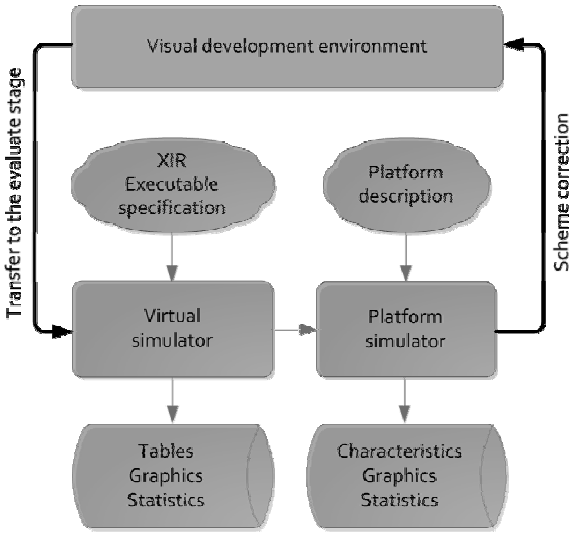

Fig. 8. Tools for early estimation and evaluation

The virtual simulator provides analysis of platform-independent characteristics of a developed program. It allows estimation of such characteristics as:

- maximal possible parallelism;
- computation space (amount of computations, used memory etc.);
- computation time;
- amount and intensity of data exchange;
- other characteristics.

B. Platform simulator

Platform simulator allows analysis of the characteristics of developed program execution on coarse-grained platform models.

Platform simulator operates with a virtual hardware platform which is a high-level representation of the hardware that is used by both hardware and software engineers. From the hardware engineer's perspective, a virtual hardware platform is a starting point for their detailed design work. It accelerates this design process. From a software engineer's perspective, a platform simulator is a simulation environment that lets them run and test their software interfaces and functionality much earlier than they could otherwise – prior to availability of the real hardware.

It allows an estimation of such characteristics as:

- requirements to embedded system cores performance;
- requirements to embedded system cores memory;
- computation cores occupation for the different allocation variants and occupation balance;
- effectiveness of hardware loading;
- bottlenecks of hardware platform, program and tasks allocation.

The coarse-grained platform model includes a description of platform structure; number, types and characteristics of computation cores; structure and characteristics of their interconnections.

A number of prototypes [14] is developed on the basis of DCNSim simulation engine [15] (Fig. 9). DCNSim engine is based on SystemC modelling language which is widely used by modelling tools for embedded systems [16, 17].

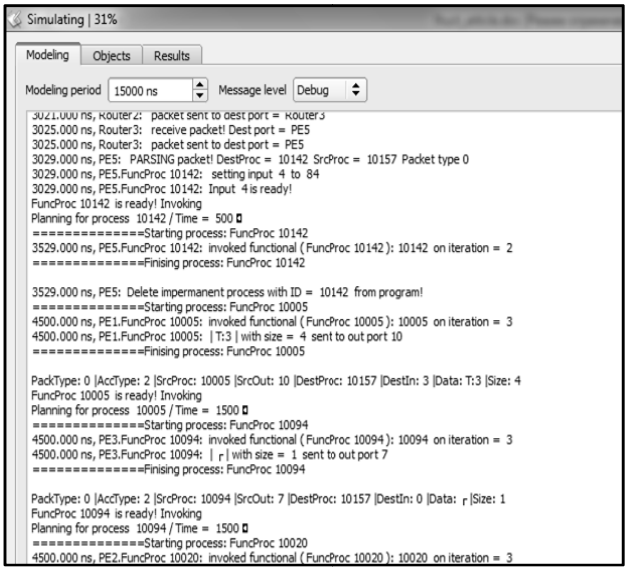

Fig. 9. Platform simulator in DCNSim simulation environment

These prototypes are fully functional. Their use allowed running an analysis of a number of use cases and getting valuable results, performing a revision and refactoring of programs. It provides a significant growth of solved tasks performance.

At that moment the opportunities of a formal model, VPL language and VIPE development environment were significantly widened. We have plans for a development of completely new simulation tools. There are several simulation engines under investigation, including perspective technology TLM 2.0 [18] among other variants.

V. INTEGRATION WITH TARGET PLATFORMS

For an integration of the development environment with a target platforms technology that uses code generators is considered.

Usage of code generation instead of code compiling is one of the crucial ideas of the proposed technology. Traditional retargetable compilers are very complex and can hardly be extended. Moreover we should generate compiled code for target platform, i.e. to make a backend we need to make a full compiler.

Instead of this, we propose constructing backend code generators. The reason is that code generators are much easier to implement than full compilers and also there are many people who know how to program a target platform and only few people who knows how to compile for a target platform. Moreover nearly all target platforms have at least one ready compiler from some language (usually platform-specific assembler, native C or platform-specific C), thus there is no real need to make another compiler for the same platform.

The technology solution provides rather simple tools of code generation into a language, which is supported by target platform compilers. Features of used formal model, the programming language and the coarse-grained approach ensure reasonable costs for a code generator development.

Code generators technology is based on coarse-grained function templates. It is the most convenient when used with coarse-grained domain libraries.

Code generator to C/C++ languages is performed by the reference generator and is included in the main toolset. Depending on a template code it produces either ANCSI C or C++ code.

Additionally this code generator can produce a parallel code based on a program scheme level parallelism. The OpenMP 4.0 technology is used as a parallel runtime engine. There are plans for support of a parallelism that is implemented using native threads of an operation system.

Templates technology ensures a portability of the application task solutions to a wide range of target platforms without a significant changing of developed schemes.

Additionally to the base code generator, the prototypes of code generators for the platform supporting MPI, MIPS assembler and DSP assembler [19] were developed under the approbation studies.

VI. Conclusions

The proposed technology ensures a support of a whole embedded software lifecycle, from an algorithm design to a deployment to an embedded system.

The formal model based development ensures the equal execution of a debugged program in any runtime environment.

An algorithm is created and debugged once. A program scheme allows saving algorithmic results and transferring embedded software to a more relevant hardware platforms within a reasonable time and with reasonable efforts. There is no need on rewriting code for each platform.

Technology guarantees rather easy way of introducing new platforms and also enables an interaction with additional instruments for functional capabilities extension.

Our plans for prospective elaborations are:

- Formal verification of program schemes.
- Scheme optimizations for a platform granularity (aggregation, allocation, interactions).
- Support of heterogeneous embedded systems (hardware units, accelerators, DSP).

Acknowledgment

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

References

[1] Joachim Keinert, Jürgen Teich. *Design of Image Processing Embedded Systems Using Multidimensional Data Flow.* New York, Springer, 2011.

[2] N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, W. Luk. A *Reconfigurable Platform for Real-Time Embedded Video Image Processing. Proceedings of 13th International Conference,* FPL 2003, Lisbon, Portugal, 2003.

[3] Michael P. Georgeff, Francois Felix Ingrand. Decision-making in an embedded reasoning system. *In Proceedings of the 11th international joint conference on Artificial intelligence (IJCAI'89),* Vol. 2. USA, 1989, pp. 972-978.

[4] Song Z1, Ji Z, Ma JG, Sputh B, Acharya UR, Faust O. *A systematic approach to embedded biomedical decision making.* Computer methods and programs in biomedicine, 11/2011; 108(2). 2011, pp. 656-664.

[5] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, G. Karsai, *Composing Domain-Specific Design Environments.* IEEE Computer, 2001.

[6] J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, B. Ellis, VEST: An Aspect-based Composition Tool for Real-time Systems, in: *Proceedings of the IEEE Real-time Applications Symposium. IEEE*, Washington, DC, 2003

[7] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, V. Prasad, Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems, in: *Proceedings of the 25th International Conference on Software Engineering,* Portland, OR, 2003.

[8] Michelle Lange, Bill Chown. *Communicate, execute, and translate – oh my! The power of xtUML and virtual platforms.* Mentor Graphics, 2013.

[9] Ivanov V., Sheynin Y, Syschikov A. Programming model for coarse-grained distributed heterogeneous architecture *Proceedings of the XI Symposium on the issue of redundancy in information systems* /Edited by prof. Kruk E.A. – SPb.: SUAI, 2007, p.246-250.

[10] Y.E. Sheynin, *"Formal model of dynamic parallel computations in parallel computing systems of experimental data processing"*, Scientific Instrumentation, 1999, vol. 9, #2. c.22–29.

[11] Sedov Boris, Alexey Syschikov, Vera Ivanova. Integrated development environment for visual parallel programming. 10th Conference of Open Innovations Assosiation FRUCT and 2nd Finnish-Rusian Mobile Linux Summit: *Proceedings printed by State University of Aerospace Instrumentation (SUAI).* 2011. Pp. 131-135.

[12] Vera A. Ivanova. Methods of domain-specific languages development // Scientific session of SUAI: *Proceedings: In 3 parts. Part I. Technical sciences* / SPb SUAI. SPb., 2014. Pp. 113-116

[13] John Vargas. *Reduce project schedules and increase quality using model driven development for design,* verification and test. Michigan, Troy, 2013.

[14] Orlov Aleksandr, Alexey Syschikov. High-Level System-on-Chip Simulator. *11th Conference of Open Innovations Association FRUCT: Proceedings printed by State University of Aerospace Instrumentation (SUAI*), 2012. Pp. 27-35.

[15] Sheynin Y.E., Volkov P.L., Onischenko L.V., Razhivin D.B., Cherniy A.S., Eganyan A.V., Nikolsky V.F., Kosyrev S.A., "Software support of the VLSI family "Multicore-designer" for the construction of the parallel structures and distributed signal processing systems", "Questions of Radio electronics", a series of "Electronic computing equipment (EWT)", issue 3, 2008.

[16] S. Balandin, M. Gillet, I. Lavrovskaya, V. Olenev, A. Rabin, A Stepanov, "Co-Modeling of Embedded Networks Using SystemC and SDL", *International Journal of Embedded and Real-Time Communication Systems (IJERTCS*), Vol. 2 (1), pp 24-49. January-March 2011.

[17] Frank Ghenassia, T*ransaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems.* Springer, 2005.

[18] Brian Bailey, Felice Balarin, Michael McNamara, Guy Mosenson, Michael Stellfox, Yosinori Watanabe. TLM-Driven Design and Verification Methodology. Cadence, 2010.

[19] Bukharenko Nikita, Syschikov Alexey. Code-generator of parallel assembly code for digital signal processor. *11th Conference of Open Innovations Association FRUCT: Proceedings printed by State University of Aerospace Instrumentation (SUAI),* 2012. Pp. 136-143.