DOI: 10.17587/mau.23.563-569

M. V. Gordin, gordinmv@bmstu.ru, **G. S. Ivanova,** gsivanova@bmstu.ru,
A. V. Proletarsky, pav@bmstu.ru, **M. V. Fetisov,** fetisov.michael@bmstu.ru,
Science and Educational Center "Robotics" Bauman Moscow
State Technical University, Moscow, 105005, Russian Federation

Corresponding author: **Fetisov Mikhail V.,** Senior Lecturer, Department of Computer Systems and Networks, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation, e-mail: fetisov.michael@bmstu.ru

Accepted on August 5, 2022

Adaptive Modelling System as a Unified Platform for Industry-Specific CAD Systems

Abstract

The risks associated with the isolated design of complex software systems within individual industries are analyzed, where not only the same thing is often done, but also the quality of the design suffers due to incomplete competence of the implementers. The approach of dividing competence and responsibility in complex software development by introducing an additional domain-specific layer of interaction between the software developer and the subject area specialists is discussed. The use of an adaptive modelling system as a tool for such separation is proposed. It is shown that the use of adaptive modelling as a common development platform for industry-specific CAD will not only improve the quality of production design in different industries, but will also simplify the design of production in related fields. Finally, it is shown that the use of a common platform will avoid the costs associated with the trend towards simplification and atomization of software developed in our country in the face of sanctions and the degradation of global connections.

Keywords: *production process modelling, production process design, modelling simulation, subject area, modelling system, adaptive modelling system, domain-specific language, problem-oriented language*

For citation: **Gordin M. V., Ivanova G. S., Proletarsky A. V., Fetisov M. V.** Adaptive Modelling System as a Unified Platform for Industry-Specific CAD Systems, *Mekhatronika, Avtomatizatsiya, Upravlenie*, 2022, vol. 23, no. 11, pp. 563—569.

DOI: 10.17587/mau.23.563-569

УДК 004.434 + 004.942:658.5

DOI: 10.17587/mau.23.563-569

М. В. Гордин, канд. техн. наук, и. о. ректора, gordinmv@bmstu.ru,
Г. С. Иванова, д-р техн. наук, проф., gsivanova@bmstu.ru,
А. В. Пролетарский, д-р техн. наук, проф., декан факультета "Информатика и системы управления", pav@bmstu.ru,
М. В. Фетисов, ст. преподаватель, fetisov.michael@bmstu.ru,
МГТУ им. Н. Э. Баумана, Москва

Адаптивная система моделирования как единая платформа отраслевых САПР

Анализируются риски, связанные с изолированным проектированием сложных программных систем внутри отдельных отраслей промышленности, при которых нередко не только делается одно и то же, но также страдает качество разработки из-за неполной компетенции исполнителей. Рассматривается подход разделения компетенции и ответственности при разработке сложного программного обеспечения за счет введения дополнительного предметно-ориентированного слоя взаимодействия разработчика программного обеспечения со специалистами предметной области. Предлагается использование адаптивной системы моделирования в качестве средства такого разделения. Показывается, что использование адаптивной системы моделирования в качестве общей платформы разработки отраслевых САПР не только повысит качество проектирования производств в различных отраслях, но также упростит проектирование производств в смежных областях. Наконец, показывается, что использование общей платформы позволит избежать издержек, связанных с тенденцией упрощения и атомизации программного обеспечения, разрабатываемого в нашей стране, в условиях санкций и деградации глобальных связей.

Ключевые слова: *моделирование производственных процессов, проектирование производственных процессов, имитационное моделирование, предметная область, система моделирования, адаптивная система моделирования, предметно-ориентированный язык, проблемно-ориентированный язык*

Introduction

At the moment, humanity is witnessing a globalisation that only recently reached its highest level in human history begin to weaken and roll back [1]. The economic relations between regions and countries that have been formed over centuries are being destroyed, sanctions are being imposed and logistical links are being disrupted. And while the COVID-19 pandemic was previously blamed, now that its influence is waning and the crisis is only worsening, there is reason to believe that the cause of degradation lies deeper and that the process of global decay may be quite profound [2].

The more complex the production, subject area or industry to be automated, the more resources are required to develop and maintain the automation system. In order not to lose control of the process, hierarchical automation clusters are formed in which competences and responsibilities are shared between the performers. The correct division of labour ensures an optimum production and support structure. Organisations that have been able to create such an optimal structure appear to be the most efficient, have less overhead to maintain internal processes and, as a result, make more profit. And so they are the ones who survive. Adam Smith made this point as early as the 18th century [3]. The collapse of the global order disrupts these structures, which means that the ability to develop and maintain complex productions, including complex modelling systems, will be greatly hampered.

Quite often, when there is an urgent need for import substitution of a product, especially in a time pressure situation, an interested party will simply copy the product. This is the best case outcome. Most often, a functionally limited version is created, and it takes a long time to reach a suitable level of quality. As a result, different industries, corporations, even businesses within industries and corporations, create their own versions of the product, with little or no compatibility between each other.

Unfortunately, this is the trend in the development of industry-specific CAD in our country at present. There are many reasons for this. Let us consider, in our opinion, the main ones.

Firstly, there is no overall management of the development process of these systems, which could coordinate the development of programs by dividing responsibilities into areas of competence. For example, the practice in the Soviet Union was to designate a research institute as the main developer of software for a particular area, and to coordinate the finalization and application of the developed programs.

Secondly, more often than not, companies interested in import substitution of their CAD systems either develop them themselves, having high competence in their subject area but low in software development, or (much less frequently) outsource development to professionals who have competence in software development but no competence in the subject area. The latter option requires well-organized client-executor interaction in the development process, which is not always possible. The result is either a functionally complete product with poor code quality, which is usually brought to an acceptable state, but further development becomes difficult, or a product of more or less quality in terms of the software component, but which does not fully solve the customer's tasks and also requires revision.

This paper proposes an approach to the allocation of responsibilities according to the competencies of the participants, not only to enable the development of software of high quality in all respects, but also to enable the development of a product that has the potential to outperform its substitute analogues by a wide margin.

Separation of competence and responsibility

Fig. 1 shows the schemes for abstract development with incomplete competencies described above.

This diagram is based on a UML (Unified Modeling Language) notation variant diagram [4]. The system boundary has been removed, and instead the competence boundaries of the participants are shown as rectangles with rounded corners. The competence boundaries circumscribe both the participant and the action (function) within that competence and for the performance of which the participant is responsible.

Fig. 1 shows that both options are incomplete in terms of the competence of the participant in relation to the function they perform — these functions are outside their competence. To solve the problem of incomplete competence there is a simple trick: allocating an additional function for which both participants have competences.

Fig. 2 shows the competence diagram, which introduces the function of forming domain-specific tools — software components that, on the one hand, simplify the work of a specialist in developing a model of their subject area and, on the other hand, allow this work to be done autonomously from the software developer. It is now the responsibility of each participant to carry out their own work within their area of expertise.

The action of "developing domain-specific tools" is within the area of competence and responsibility

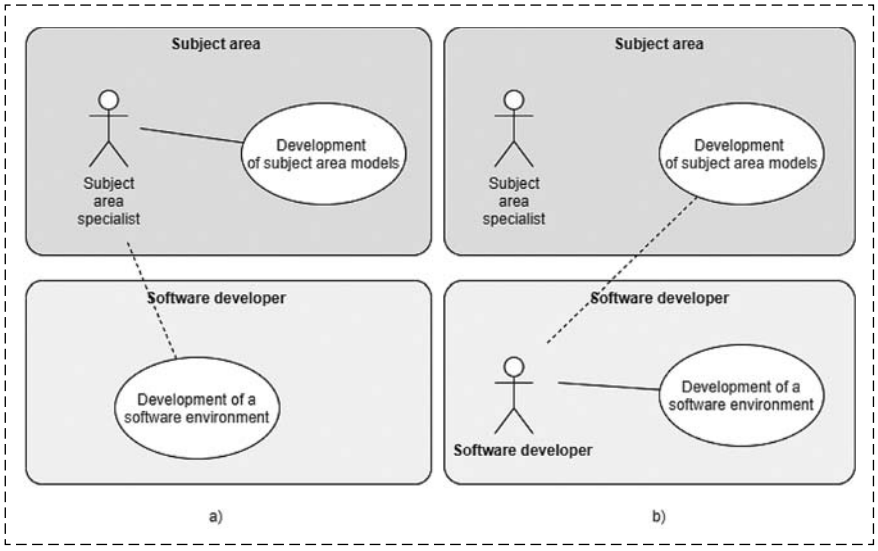

Fig. 1. Schemes with incomplete competence: incomplete competence in software development for a subject area specialist (*a*); incomplete competence in developing models for a subject area for a software developer (*b*)

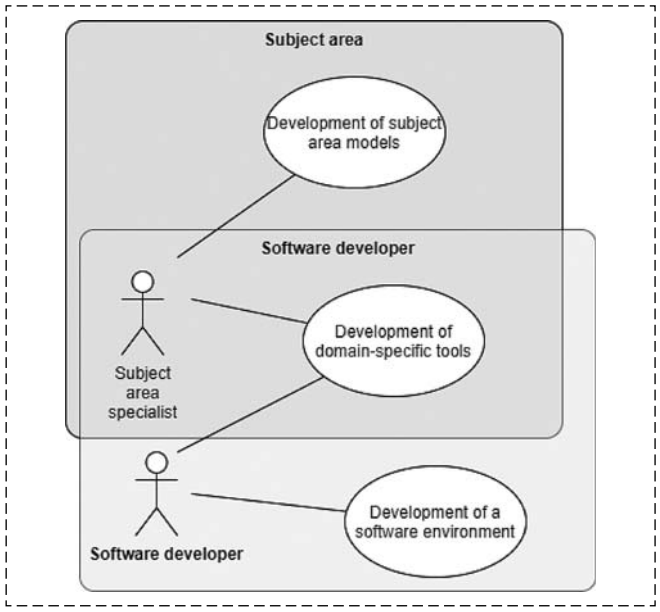

Fig. 2. Scheme of division of responsibility according to competences

of both participants in the work. This means they must somehow separate and regulate responsibilities within this function.

This technique is widely used in information technology, but it is often forgotten when planning the implementation of complex software systems. However, it is in the development of complex models that the division of competence and responsibility is most effective.

Adaptive modelling system

The Adaptive Modelling System (AMS) provides an original and unique toolkit that allows the development of domain-specific languages (DSL) that are more natural for experts in the respective subject areas to describe their domain models [5].

Fig. 3 (see the second side of the cover) shows the interface of the AMS when working with a model described by a system of ordinary differential equations, in the example, the Lorentz attractor is presented. It can be assumed that both DSLs for describing dynamical systems and visualization tools that allow the construction of graphs and state change diagrams of a dynamical system can be used to describe many aspects of modelling more complex systems in various subject areas.

This screenshot shows the operation with a prototype AMS developed at the Faculty of the CS6 at the Bauman Moscow State Technical University 6].

Separately, it is important to note that an AMS should have an open software architecture in order to minimize the risks associated with the development and maintenance of DSLs and visualization tools [7]. Fig. 4 shows a diagram of the system components to allow for extensibility of the AMS architecture.

Let us take a closer look at some of the components.

Unified means of working with DSL. This component is the service in which the AMS business logic is implemented. This service provides several APIs (Application Programming Interface):

The DSL API is a set of DSL interfaces that allow you to load extensions, grammars, perform syntactic and semantic analysis (including providing syntax highlighting information, remarks and errors), run the model for execution, debug the model, etc.

The BNF & Interpreter API is a set of interfaces provided to connect extensions and support the interpretation of the DSL.

The Module API is a set of interfaces that allows modules implemented in compiled languages (C, C++, FORTRAN, etc.) to be connected for use in the underlying AMS language and in the DSL.

Integrated development environment. This component uses the "Unified means of working with DSL" service to generate models, run them, debug them, etc., and also provides a Visualizer API that allows the connection of external graphical extensions.

External graphical extensions should allow, for example, forming the text of a model from a graphical

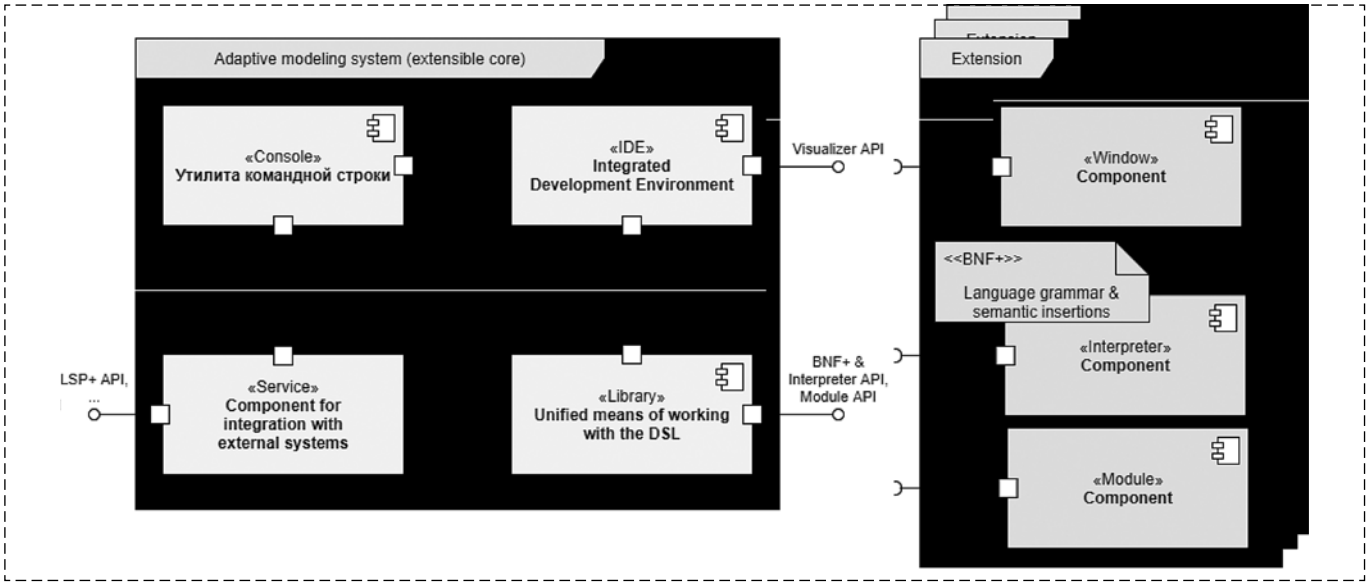

Fig. 4. Diagram of the components of the AMS open software architecture

constructor and interacting with it. For example, you can automatically generate a diagram of an algorithm or assemble an algorithm from the graphical elements of a flowchart, which can be very visual, including when debugging the application, and useful in training.

The imperative of a domain-specific language

The importance and primacy of the use of DSL in the development of the final product of a subject area is due to the several advantages that this approach offers. Among these advantages is the ability to use versatile visualization tools, as described above.

Another advantage is the ability to easily work collaboratively with version control tools and code repositories, as well as using modern tools such as continuous integration and many others associated with it.

Parallel to the development and maintenance of DSLs, visualisation tools can be developed and maintained based on the code structure written using the DSL. That is, the code is the primary description of the subject area, and schemes and diagrams are the graphical interpretation of the code. Moreover, a schema can be edited, which will be reflected in the code structure. Such a division is used in many subject areas and has shown its consistency, having stood the test of time.

Fig. 5 and Fig. 6 (see the second side of the cover) show examples of schematics when building an image processing structure using shaders in Blender [8], as well as a schematic description of an electrical circuit using the System Verilog [9]. Behind a cowl of work with the models in these systems the primary domain-specific language of the description of these models hides. At the same time, the given schemes are iterative, allow changing parameters of elements, running and debugging of models.

As another example of a successful implementation of the separation of the linguistic and graphical components of modelling, universal modelling systems such as MATLAB, Modelica and many others, which have a basic language name in their names, but are often associated specifically with graphical tools that allow visualization of models.

It is also important to note a feature of the language implementation of a model, such as its great generality. In a domain-specific language, it is usually possible to describe a complex model that cannot be represented in graphical form. This diversifies the process of developing complex models.

Modern code editing tools allow one to perform not only lexical and syntactic analysis and highlight structure or errors directly when typing or modifying code, but also to perform semantic analysis from a domain-specific perspective, which leads to the identification of a certain class of problems at a very early stage of model formation [10].

Division of competence when working with industry-specific CAD

Let us return to competence diagrams. Fig. 7 shows a competency diagram for the use of an AMS in a specific domain.

This diagram has more participants and more functions that are specific to the implementation of

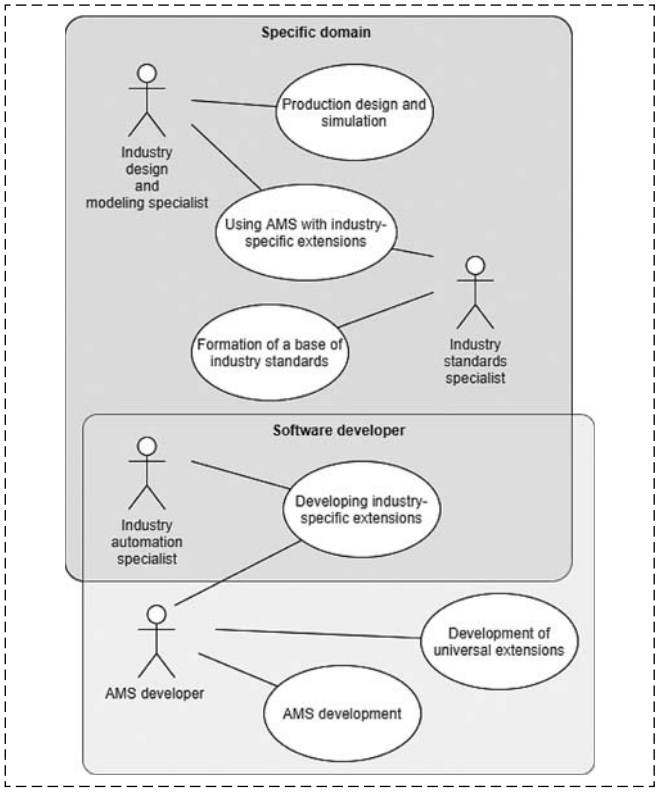

Fig. 7. Using an AMS in production design and modelling

CAD for a particular industry. Let us take a more detailed look at the diagram.

The AMS developer is responsible, within his area of expertise, for the development of the AMS, universal solutions, and shares the development of industry extensions with the industry automation specialist. Next, the industry standards specialist is involved, using the AMS and industry-specific extensions to build the industry standards base. Finally, the industry design and simulation specialist performs production design tasks using the AMS, the industry extensions and the standards database.

The diagram shown in Fig. 7 is an example, in a particular case there may of course be different layers and sequences of interaction. In this case it is important to show how the use of the division of competence and responsibility can work in complex domain-specific, simplifying the interaction between participants and ultimately improving the quality of their work.

As shown, the AMS developer generates some universal extensions that can be used in different subject areas and industries. Fig. 8 shows a diagram of AMS components, both with universal extensions and with extensions for domain-specific.

Responsibility management in multi-domain-specific cooperation

Using an AMS as a common platform for the development of industry-specific CAD and simulation systems has a very important advantage: the compatibility of extensions for different subject areas. This makes it possible to build models using extensions from related industries.

Fig. 9 shows a diagram of the division of competences in a two-domain cooperation. Of course, there may be other options for cooperation, including more complex and multi-domain cooperation.

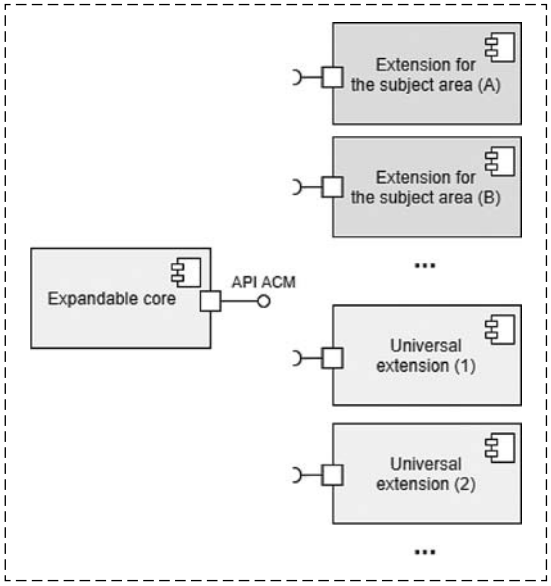

Fig. 8. Component enlargement diagram of AMS components with universal extensions and domain-specific extensions

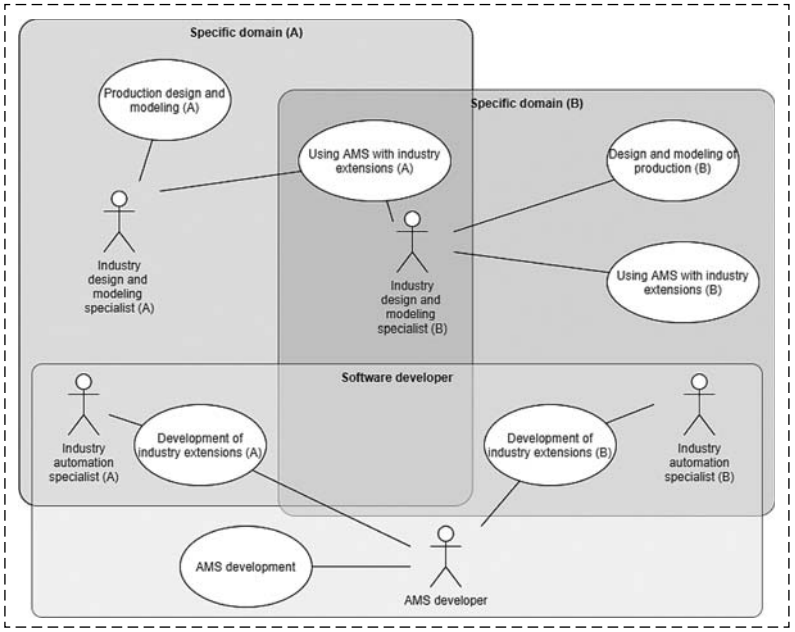

Fig. 9. Using industry extension (*A*) in industry design and modelling (*B*)

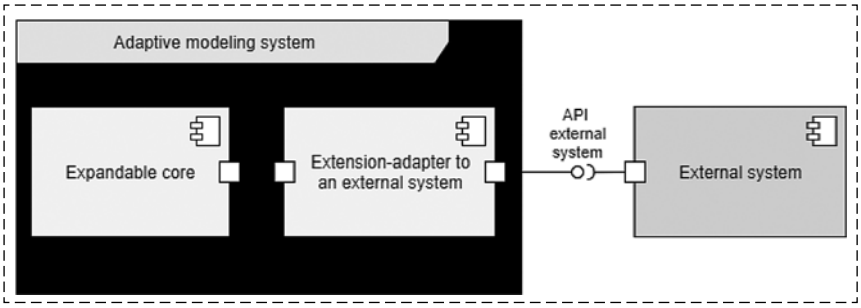

Fig. 10. Connecting an external system via an extension adapter

Possibility of connecting external systems

What if there is some modelling or design system, developed independently of the AMS, that needs to be used to extend the production model. In this case, much depends on the software architecture of the external solution.

Fig. 10 shows a component diagram with an extension-adapter, which allows an external system to be connected to the AMS via its API and to do design work together. However, the feasibility of such a solution is highly dependent on the implementation of the specific external software system.

Analogues of an adaptive modelling system

The main function of an AMS is to be able to work with many domain-specific languages built on the basis of a base language. Thus, the set of DSL in AMS is a hierarchy of dialects of the basic language that have semantic features, i.e. extending the basic operational semantics. In order to optimally implement this approach, substantial unification of lexis, syntax and semantics within AMS is used. Such unification largely contradicts the classical notion of compiler construction. For example, the implementation of the built-in generator of compilers on the basis of the common parsing automaton for automatically generated grammar tables [7] was criticized by Nicholas Wirth in his work "Building compilers" [11] and suggested using common external generators, e.g., YACC [12], Bison [13] or others. However, in the context of a compact description of a potentially large number of DSL dialects and their frequent refinement, the canonical approach seems unnecessarily cumbersome.

Basically, for this reason there are no direct counterparts to AMS. However, there are some developments in which similar ideas and trends can be seen.

For example, the Russian system **AnyLogic** [14] is software for simulating business processes in logistics, healthcare, manufacturing and banking, as well as any other processes that can be represented as a sequence of operations. The tool has a modern graphical interface and allows the use of Java language for model development.

We can also note the trend towards unification of compiler construction on the part of universal languages. Examples are **GCC** (GNU Compiler Collection) and **LLVM** (Low Level Virtual Machine). The principle of work of these program infrastructures is the following: on the frontend (the upper level of the compiler) the source program code is read, its parsing is performed and an abstract syntax tree is generated and on the backend (the lower level of the compiler) this tree is converted into some intermediate representation which is optimized and translated into an assembly language program [15]. This approach allows compilers from many programming languages to be implemented on the frontend. All these compilers form an intermediate representation and further processing is unified.

In GCC the abstract syntax tree is converted to RTL (Register Transfer Language) which is an intermediate internal representation close to an assembly language, whereas in LLVM there is an intermediate assembly language which can be transformed at compile time [16].

There are also alternative solutions for compiler and DSL implementations that offer additional features and extensions.

For example, **Xtext**, a framework for developing programming languages and DSL, allows generating not only a parser and a compiler, but also a fully customizable Integrated Development Environment (IDE). The DSL is described in Xtext with Java inserts, and the source code is created in Java [17].

Tree-sitter is a tool for creating parsers in C from a grammar description described in a language resembling RBNF (extended Backus-Naurus form). This tool allows you to build a specific syntax tree from a source file and efficiently update it when editing the source file [18]. This system is rather limited: for example, there is no possibility to put semantic inserts in the grammar description.

Domain-Specific Language Designer is a dedicated Visual Studio solution for DSL design. Microsoft offers creation and editing of DSL definition by means of graphical interface: elements and relations in the model of the subject domain are shown on the model schema. The code generator takes this definition as input and creates C# source code as output [19].

MPS (Meta Programming System) is a metaprogramming system developed by JetBrains [20]. It implements the language-oriented programming paradigm [21], is a language development environment and at the same time an IDE for languages under development.

The presented solutions are designed for designing universal programming languages without taking into account the peculiarities of a particular subject area, or for designing DSL, but with rather limited functionality. In general, the prevailing view in the information technology community is that "DSL is a programming language with limited expressive power, oriented towards a specific subject area" (Martin Fowler) [22]. In reality this may not be the case. A DSL in an adaptive modelling system is a programming language which extends the semantics of a basic generic language. More often than not, the semantics of the underlying language will be restricted in a particular DSL, but this is not mandatory. The basic AMS language, in turn, should be constructed in such a way that its semantics can be easily extended — a very interesting, but far beyond the scope of this article, topic for discussion.

Ideas related to the implementation of a modelling system that allows combining different subject areas are also emerging in the Western scientific community. Thus, a similar approach in decision-making systems is proposed, which may be useful for use in AMS as well [23], [24].

Conclusion

The use of an adaptive modelling system as a single platform for branch CAD will not only provide a high-quality software component for automating the domain-specific by dividing the competence and responsibility of the implementers, but will also make it easier and faster to design complex production at the interface between industries.

The use of AMS will also avoid the costs of designing complex productions associated with the trend towards simplification and atomization of software developed in our country, in the face of sanctions and the degradation of global connectivity.

References

1. **Glazyev S.** The last round of liberal globalization [Electronic resource], available at: https://izborsk-club.ru/11870 (access date: 07.07.2022) (in Russian).

2. **Smirnov G.** The Undefiled Ways of Globalisation [Electronic resource], available at: https://www.kommersant.ru/doc/5295109 (access date: 07.07.2022). (in Russian).

3. **Smith A.** (1776). An Inquiry into the Nature and Causes of the Wealth of Nations [Electronic resource], available at: https://books.google.ru/books?id=C5dNAAAAcAAJ (accessed 07.07.2022).

4. **Unified** Modeling Language [Electronic resource], available at: https://www.omg.org/spec/UML (accessed 07.07.2022).

5. **Ivanova G. S., Zhiltsov A. I., Fetisov M. V., Chulin N. A., Yudin A. E.** Adaptive Modelling System, *Automation. Modern Technologies*, 2020, no. 11, pp. 500.

6. **SIMODO** in Bauman Moscow State Technical University repository [Electronic resource], available at: https://bmstu.codes/lsx/simodo (access date: 07.07.2022).

7. **Ivanova G. S., Fetisov M. V., Malkina T. A., Raldugina A. V.** Unification of work with subject-oriented languages and open software architecture in adaptive simulation system, *Dynamics of Complex Systems*, 2021, vol. 15, no. 3, pp. 36—47.

8. **Blender** [Electronic resource], available at: https://www.blender.org (accessed 01.06.2021).

9. **1800-2017 — IEEE** Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language [Electronic resource], available at: https://ieeexplore.ieee.org/document/8299595 (accessed 07.07.2022).

10. **Ivanova G. S., Fetisov M. V.** The concept of contract management in the base language of the adaptive modeling system, *3rd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA),* IEEE, 2021, pp. 833—836.

11. **Wirth N.** Building compilers, Moscow, DMK Press, 2016 (in Russian).

12. **BYACC** — Berkeley Yacc — Generate LALR(1) parsers [Electronic resource], available at: https://invisible-island.net/byacc/byacc.html (accessed 04.11.2019).

13. **GNU** Bison [Electronic resource], available at: https://www.gnu.org/software/bison (accessed 04.11.2019).

14. **AnyLogic** [Electronic resource], available at: https://www.anylogic.ru (accessed 01.06.2021).

15. **Griffiths A.** GCC. A handbook for users, programmers, and system administrators, Diasoft. 2004.

16. **Brown A., Wilson G.** LLVM (Chris Lattner), The Architecture of Open Source Applications, 2011.

17. **Xtext** [Electronic resource], available at: https://www.eclipse.org/Xtext/index.html (accessed 07.07.2022).

18. **Tree-sitter** [Electronic resource], available at: https://tree-sitter.github.io/tree-sitter (accessed 07.07.2022).

19. **Domain-Specific** Language Designer [Electronic resource], available at: https://docs.microsoft.com/ru-ru/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages (accessed 07.07.2022).

20. **MPS** — Meta Programming System [Electronic resource], available at: https://www.jetbrains.com/ru-ru/mps (accessed 07.07.2022).

21. **Dmitriev S.** Language-oriented programming [Electronic resource], available at: http://rsdn.org/article/philosophy/LOP.xml (access date: 07.07.2022).

22. **Fowler M., Parsons R.** Domain Specific Languages, ADDISON-WESLEY, 2010.

23. **Roci M., Salehi N., Amir S.** et al. Towards circular manufacturing systems implementation: A complex adaptive systems perspective using modelling and simulation as a quantitative analysis tool, *Sustainable Production and Consumption,* 2022, vol. 31, pp. 97—112.

24. **Roci M., Salehi N., Amir S.** et al. Multi-method simulation modelling of circular manufacturing systems for enhanced decision-making, *MethodsX*, 2022, vol. 9, article 101709.