Automata-Based Programming and Automata-Based Control

Anatoly Shalyto

St. Petersburg, St. Petersburg State University of Information Technologies, Mechanics and Optics,
e-mail: shalyto@mail.ifmo.ru

Abstract: The article contains main theses of automata-based programming and discusses its advantages when applied in software engineering. The apparatus for automata-based programming is described. Automata-based control is the core of automata-based programming.

1. INTRODUCTION

Most expert programmers reckon that the software engineering industry has no serious problems. Pretended absence of problems leads to the situation when software engineering solutions are mostly ad hoc, based on the programmers' experience. All difficulties are considered "the inevitable evil of the industry". The failure of the major portion of software projects doesn't influence the opinion of the majority.

Software engineering theorists have a fully opposite opinion – in 1968 they "openly admitted the software crisis" (see Dijkstra [1972]). Nowadays, this thesis is sometimes disputed, e. g. professors N. Wirth and J. Gutknecht claimed (when visiting SPbSU ITMO) that they don't see problems in software engineering except for the area of programming drivers with complex behavior (note that implementation of such behavior is the topic of the current article).

In spite of such views of influential scientists, many theorists think that the mentioned crisis still goes on. They tried resolving the crisis by switching from "the art of computer programming" (see Knuth [1997]) to software engineering (see Software Engineering [1968], Software Engineering techniques [1969]) which is actively promoted by B. Meyer, the "successor" of N. Wirth.

The continuing crisis is connected to the fact that software engineering specialists almost never use approaches developed in other areas. There is an opinion (see Kai-Yuan C. et al. [2002]) that software engineering should use the experience of designers of automated control systems, and it would be advisable to make a step backward and turn to the founders of cybernetics, such as N. Wiener, J. von Neumann, and W. Ashby.

The stated above is advocated by the specialists in Software Cybernetics (see Kai-Yuan C. et al. [2002]), the area in which the first international workshop was held in 2004 and then became annual.

In this article we state the bases of automata-based programming, the approach being developed since 1991 (see Shalyto [1991]). The research in the area of automata-based programming refers both to software engineering and software cybernetics. The ideas come from the automata theory and control theory – two of the three components of cybernetics (the third being information theory). We emphasize that automata-based programming doesn't mean programming with the use of automata, but the entire programming paradigm and programming technology aimed for designing systems with complex behavior (see Shalyto [1998]). For comparison, UML (Unified Modeling Language) is just a notation while RUP (Rational Unified Process) is the process that uses the notation (see Booch et al. [2005]).

The proposed approach is close to the approach of Harel [1992], which was described by F. Brooks to be possibly revolutionary (see Brook [1995]).

2. AUTOMATA-BASED PROGRAMMING AS A PROGRAMMING STYLE

Automata-based programming is one of programming styles (see Nepeyvoda [2005]). This term was proposed in Shalyto [1998].

To put it simple, the approach proposes to **describe the behavior of programs using automata which are later isomorphically converted into code**.

Automata have been successfully used in software in the field of compilers, protocol implementations etc. In Shalyto [1998] it was suggested to use automata not as a discrete math objects but as **the universal approach for implementing programs with complex behavior**, especially the reactive ones (see Harel et al. [1990]).

Note that "programming using automata" can't be seen as a programming paradigm, as it still leaves unclear how to design and implement a program as a whole using automata.

3. AUTOMATA-BASED PROGRAMMING AS A PROGRAMMING PARADIGM

Many systems with meaningful behavior are nothing but automated objects.

Automated control object is the aggregate of the *control object (CO)* and *control system (CS)* related with feedbacks (Figure 1).

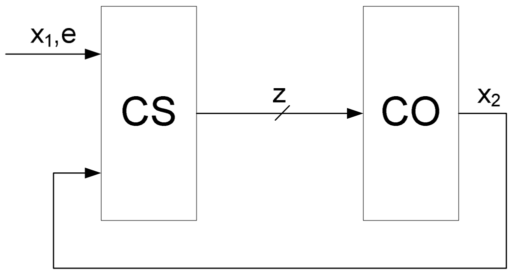
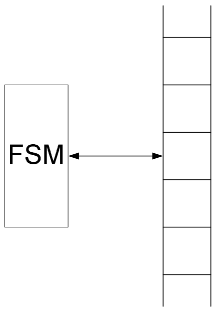


Fig. 1. Automated control object

The task of designing automated control objects is discussed in every course of automated control theory. Surprisingly, it didn't affect software engineering practice, in spite of the fact that one of the major models in algorithms is the Turing machine (Figure 2).



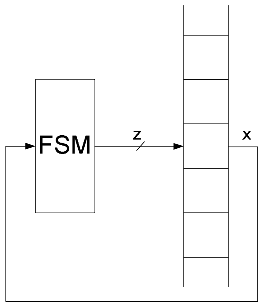


Fig. 2. Turing Machine **Fig. 3.** Turing machine as automated control object

Turing machine is basically an automated object (see Shalyto, Tukkel [2002a]) in which the control system is a finite state machine (an automaton) and the control object is the infinite tape (its cells) (Figure 3).

Switching from Turing programming to practical programming is carried out due to complication of complex object, which can run any complex operations. Meanwhile the control system is a system of interacting automata.

Thus, the proposed approach is a generalization of Turing machine that allows implementation of arbitrarily complex algorithms. Meanwhile, the theoretical results about the limitations on recognized languages (e. g., regular languages for finite state machines (FSM) and context-free languages for FSM with stack) can be ignored, since **the proposed approach concerns not automata, but automated control objects in general,** in which the complexity of the control objects can be arbitrarily high.

The main feature of automata-based programming is the following: **programs should be coded in a way similar to the automatization of technological processes**.

Based on domain analysis, one defines the input action sources, the control system and the control objects. Here a control system is a system of interacting automata. The control objects implement output actions and form feedback to the control system.

All listed components are shown on a *relation diagram*, which can also be an *interaction diagram*. For each input and output action, the full name is shown as well as a short identifier, that is used in *transition diagrams* (*state diagrams*) and in the code. Short identifiers make transition diagrams compact and comprehensible.

Control objects can be either real or virtual (implemented as programs). In the first case, their logic is fixed, in the second case the logic of the control objects can and should be extracted into the automata in the control systems.

Paradigm of automata-based programming is **representation and implementation of programs as systems of automated control objects**.

4. MAIN THESES OF AUTOMATA-BASED PROGRAMMING

The main concept in the automata-based programming is *state* (see Shalyto [1998]). In automata-based programming one can distinguish between two types of states – control and computational. As in Turing machines, a few control states is enough to control many computational states (see Shalyto, Tukkel [2002a]). In the following text, only control states will be mentioned.

Automata can be abstract (input and output actions are formed consequently) or structure (actions formed simultaneously). In automata-based programming the structure automata are used.

Time is not used in automata. If needed, delay elements are added as control objects. This way, the time elapse events are received by the automata as input actions, so-called *timed automata* (see Shalyto [1991] and Allur, Dill [1994]).

When designed manually, automata should possess *cognitive properties*, which are reached when automata are represented as transition graphs (transition diagrams). If generated automatically, automata can be represented as a table, which is less comprehensible.

5. ADVANTAGES OF AUTOMATA-BASED PROGRAMMING

In the proposed approach it is supposed that the code generation starts only after the program being designed. Note that **in engineering each project always ends with issuing**

the project documentation. Unfortunately, in traditional software engineering this is not the case. In automata-based programming we propose that each project must contain not only user manual (which is usual for software projects), but the project documentation including relation diagrams and transition graphs for each automaton.

The manually designed automata are **formally** and **isomorphically** transformed into code (manually or automatically) which either works right away or needs only minimal debugging.

As the logic is represented in visual form instead of usual text form, making fixes is simpler and it is much easier to understand the logic for people other than the author, as well as by the author after some time passes (which can be a problem with traditional code).

The next advantage is the possibility of effective verification of automata-based programs using Model Checking (see Clarke et al. [1999]).

Finally, automata-based programs are naturally parallelized which is important for multi-core processors.

6. TYPES OF AUTOMATA-BASED PROGRAMMING

Automata-based programming is developing in three main directions: logical control, state-based programming and state-based object-oriented programming.

Logical control tasks are those that have binary input and output variables. Here automata replace logical schemes, which is quite natural.

State-based programming concerns reactive systems, e. g. majority of embedded systems. Reactive systems are more complex than logical control systems, for the following reasons:

- Input actions are not only input variables but also events;
- Programs are executed on events, not in a cycle;
- Output actions can be not only binary, but more complex (*hybrid automata*, see Alur et al. [1995]);
- Automatons can contain not only nested states (see Harel et al. [1990]) but also nested automata;
- Automata can interact not only via checking state numbers (as in logical control systems, see Shalyto [1998]), but also via nesting, executability and event/message exchange.

Objected-oriented state-based programming allows different solutions. First, automata can be used as class methods or classes. A deeper interosculation between object-oriented programming and automata is possible thanks to the practice of using patterns, such as *State* pattern and *State Machine* pattern. Other approaches for combining object-oriented and automata-based paradigms exist and are classified in Naumov et al. [2005].

7. VERIFICATION OF AUTOMATA-BASED PROGRAMS

Automata-based programs are subject to verification by Model Checking. The reason is that the behavior model of the program and its description is either the same or can be transformed to each other automatically, which is impossible for traditional programs. There is no semantic gap between program requirements and the model, thus the automata-based programs are checkable by definition (see Velder, Shalyto [2007], Kuzmin, Sokolov [2008], and Egorov, Shalyto [2008]).

This makes the proposed approach applicable for responsible systems. We hope that in future technical requirements for software in such areas will require using automata-based programming.

8. AUTOMATIC GENERATION OF AUTOMATA-BASED PROGRAMS.

The main effort in creating an automata-based program is designing the automata. There exist problems which can be solved using automata, but the desired automata are hard to design manually and heuristically. There are formalized methods for designing automata. Dynamic programming is used in Orshanskiy, Shalyto [2007], but its use is very limited. An approach that turns out to be more universal is genetic programming (see Lobanov, Shalyto [2007] and Davydov et al. [2008]).

9. AUTOMATA-BASED PROGRAMMNG TECHNOLOGY

A programming technology (including all lifecycle phases) was developed for the automata-based programs. The technology is described in Shalyto [1998, 2000a, 2000b, 2001] and Shalyto, Tukkel [2001], and expounded for the general public in Naumov, Shalyto [2003].

10. APPARATUS FOR AUTOMATA-BASED PROGRAMMING

It is possible to generate code that is isomorphic to a transition graph of an automaton. In Goloveshin [2002] a tool is described that allows generating the code automatically. It uses `switch` operator of the *C* programming language. This process was generalized in Kanzhelev, Shalyto [2006] where it is shown that a similar approach can be used for an arbitrary programming language. The tool that implements this approach is *MetaAuto*.

A powerful tool is *UniMod* (see Gurov et al. [2005, 2007]), that automatizes the process of designing object-oriented automata-based programs. In this tool, the structure of the program is represented as class diagrams, which are shown not in the traditional way, but as relation diagrams of automata with event sources and control objects. Program dynamics is described in UML which allows not only nested states but also nested automata. Input and output actions are manually coded *Java* programs that contain practically no logic. The entire system can be compiled into working code or can be run in the interpretation mode. The tool is an open source project (http://unimod.sourceforge.net/intro.html), downloaded more than 40 000 times. It supports the concept of *executable UML.*

11. DIFFERENCE FROM OTHER APPROACHES

Automata are used in software more and more often. Tools for programming using automata are developed, such as *Stateflow* (http://www.mathworks.com/products/stateflow/), extension of *MatLab* package, and *Windows Workflow Foundation* (http://itc.ua/node/23217) by *Microsoft*, in which state machines are used as nothing but the programming language.

The proposed method is different because it suggests **using automata not sometimes, but for all objects with complex behavior**. Its application doesn't depend on used software or hardware. Whereas other approaches either suggest using specific tools or give solution only to specific problems.

The proposed approach (practically) eliminates the need to debug the resulting programs. Use of the proposed approach doesn't always decrease the time of program development, compared to the traditional approach. However, the resulting programs have lots of virtues described above. It is reasonable to suppose that for responsible objects (the ones that require verification) use of automata-based programming can become imminent. The research in this direction is actively pursued (see Regan, Hamilton [2004] and Egorov, Shalyto [2008]).

12. FOUNDATION FOR OPEN PROJECT DOCUMENTATION

In systems of information control in manufacturing, it is extremely important for programs to have project documentation. A whole foundation was created in 2002 by A. Shalyto to achieve this goal (see Shalyto [2004]). As a part of this foundation, more than 110 projects made by students of Computer Technologies Department of SPbSU ITMO are published at http://is.ifmo.ru/.

13. APPLYING AUTOMATA-BASED PROGRAMMING WHEN DESIGNING INFORMATION CONTROL SYSTEMS IN MANUFACTURING

Automata-based programming is used when developing large number of information control systems in manufacturing. E. g. control system for ship diesel generator (see Shalyto, Tukkel [2002b]) and cryogen-vacuum plant (using LabVIEW package, see Vavilov [2005a]), drives (see Vavilov [2005b]) and backflush (see Vavilov [2005c]).

14. AUTOMATA-BASED CONTROL

As the highest emphasis in placed on the control, we can talk about *control paradigm* that was called "automata-based control" in Shalyto [1998]. This paradigm was approved in practice multiple times, including implementation of software for complex behavior systems, e. g. in Shalyto, Tukkel [2002b].

The use of automata in control system design was (until recently) considered in the context of hybrid automata (see Alur et al. [1993]). Extra accent for using automata was driven by the plenary lecture of R. Brokett at IFAC congress (see Brokett [2008]) on simplification of complex control systems design process.

15. CONCLUSION

The proposed approach helps improve the quality of information control systems in manufacturing because of the following reasons (Polikarpova, Shalyto [2010]):

- specification in visual form;
- formal and isomorphic transformation from the specification to the code;
- creation of protocols in terms of automata;
- verification in terms of automata;
- possible automatic generation of automata;
- possibility of creation of multi-agent systems for object with complex behavior (see Paraschenko et al. [2006]);
- project documentation for programs.

Many works on automata-based programming can be found at http://is.ifmo.ru/. This site also contains examples of practical uses of the proposed approach.

REFERENCES

Alur R., Courcoubetis C., Henzinger A., Ho P., "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Lecture notes in computer science*, vol. 736, 1993.
Alur R., Dill D., "A theory of timed automata," *Theoretical Computer Science*, 1994.
Alur R. et al., "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138 (1), 1995.
Booch G., Rumbaugh J., Jacobson I., "Unified Modeling Language User Guide, 2nd Edition," *Addison-Wesley*, 2005.
Brokett R., "Reduced Complexity control systems," *Proceedings of the 17-th World Congress The International Federation of Automatic Control, Seoul*, 2008.
Brooks F., "The Mythical Man-Month," *Addison-Wesley*, 1995.
Clarke E., Grumberg O., Peled D., "Model Checking," *The MIT Press*, 1999.
Davydov A., Sokolov D., Tsarev F., "Application of Genetic Algorithms for Construction of Moore Automaton and

Systems of Interacting Mealy Automata in "Artificial Ant" Problem," *Proceedings of the Second Spring Young Researchers' Colloquium on Software Engineering, SPbSU*, vol. 1, 2008.

Dijkstra E., "The Humble Programmer," *Commun. ACM,* vol. 15(10), pp. 859–866 (1972).

Egorov K., Shalyto A., "The method of Automata programs verification," *Information and Control Systems*, vol. 5, 2008 (in Russian).

Goloveshin A., "Converter Visio2Switch," 2002. http://is.ifmo.ru/progeny/visio2switch/

Gurov V., Mazin M., Narvsky A., Shalyto A., "Unimod: Method and Tool for Development of Reactive Object-Oriented Programs with Explicit States Emphasis," *Proceedings of St. Petersburg IEEE Chapters. International Conference "110 Anniversary of Radio Invention," SPb ETU "LETI,"* vol. 2, 2005.

Gurov V., Mazin M., Narvsky A., Shalyto A., "Tools for Support of Automata-Based Programming," *Programming and Computer Software*, vol. 33 (6), 2007.

Harel D., "Biting the Silver Bullet: Toward a Brighter Future for System Development," *Computer*, vol. 1, 1992.

Harel D. et al., "Statemate: A Working Environment for the Development of Complex Reactive Systems," *IEEE Trans. Software Eng.*, vol. 4, 1990.

Knuth D., "The Art of Computer Programming," *Addison-Wesley*, 1997.

Kuzmin E., Sokolov V., "Modeling, Specification, and Verification of Automaton Programs," *Programming and Computer Software*, vol. 34 (1), 2008.

Naumov L., Shalyto A., "Automata Theory for Multi-Agent Systems Implementation," *International Conference on "Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering", KIMAS-03, Boston: IEEE Boston Section*, 2003

Nepeyvoda N. N., "Styles and methods of programming," *Moscow: Internet-university of information technologies*, 2005 (in Russian).

Software Engineering, *Germany: NATO Science Committee,* 1968. http://www.europrog.ru/book/nato1968e.pdf

Software Engineering Techniques, *Italy: NATO Science Committee,* 1969. http://www.europrog.ru/book/nato1969e.pdf

Kai-Yuan C., Chen T. Y., Tse T. H., "Towards Research on Software Cybernetics," *Proceedings of 7th IEEE International on High-assurance Systems Engineering (HASE 2002). Los Alamitos. IEEE Computer Society Press*, 2002.

Kanzhelev S., Shalyto A., "Automatic generation of automata code," *Information and Control Systems*, vol. 6, 2006 (in Russian).

Lobanov P., Shalyto A., "Application of Genetic Algorithms for Automatic Construction of Finite-State Automata in the Problem of Flibs," *Journal of Computer and Systems Sciences International*, vol. 46 (5), 2007.

Naumov L., Korneev G., Shalyto A., "Methods of Object-Oriented Reactive Agents Implementation on the Basis of Finite Automata," *2005 International Conference on "Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering" (KIMAS-05), Boston: IEEE*, 2005.

Orshanskiy S., Shalyto A., "Using dynamic programming in solving problems with finite state machines," *Computer tools in education*, vol. 4, 2007 (in Russian).

Paraschenko D., Shalyto A., Tsarev F., "Modeling Technology for One Class of Multi-Agent Systems with Automata Based Programming," *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, 2006.

Regan P., Hamilton S., "NASA's Mission Reliable," *IEEE Computer*, January 2004.

Shalyto A., "Software implementation of control automata," *Ship-building industry. Series "Automata and remote control"*, vol. 13, 1991 (in Russian).

Shalyto A., "SWITCH-technology. Algorithmic and programming methods in solution of the logic control problems," *St. Petersburg: Nauka (Science)*, 1998. LC Control Number: 98184413 (in Russian).

Shalyto A., "Software Automation Design: Algorithmization and Programming of Problems of Logical Control," *Journal of Computer and Systems Sciences International*, vol. 39 (6), 2000.

Shalyto A., "Logic Control. Hardware and software algorithm implementation," *St. Petersburg: Nauka (Science)*, 2000. LC Control Number: 2001425055 (in Russian).

Shalyto A., "Logic Control and "Reactive" Systems: Algorithmization and Programming," *Automation and Remote Control*, vol. 62 (1), 2001.

Shalyto A., "New Initiative in Programming – Foundation for Open Project Documentation", 2004. http://www.codeproject.com/KB/architecture/nifopd.aspx

Shalyto A., Tukkel N., "SWITCH-Technology: An Automated Approach to Developing Software for Reactive Systems," *Programming and Computer Software*, vol. 27 (5), 2001.

Shalyto A., Tukkel N., "From Turing programming to automata-based programming," *PC World*, vol. 2, 2002 (in Russian).

Shalyto A., Tukkel N., "Control system for diesel generator (fragment), Project Documentation," 2002. http://is.ifmo.ru/projects/dg/

Vavilov K., "LabVIEW and SWITCH-technology. Methods of algorithmization and programming for problems of logic control," 2005. http://is.ifmo.ru/progeny/_vavilov2.pdf.zip

Vavilov K., "Programming logic controllers SIMATIC S7-200 (SIEMENS). Methods of algorithmization and programming for problems of logic control," 2005. http://is.ifmo.ru/progeny/_metod065.pdf

Vavilov K., "Programming logic controllers SIMATIC S7-300 (SIEMENS). Organization of interaction of independent local control systems using automata approach," 2005. http://is.ifmo.ru/progeny/_s7300.pdf

Velder S., Shalyto A., "Verification of simple automata-based programs using the model checking method," *Information and Control Systems*, vol. 3, 2007 (in Russian).

Polikarpova N., Shalyto A., "Automata-Based Programming," St. Petersburg: *Piter*, 2010 (in Russian). http://is.ifmo.ru/books/_book.pdf