ИСП
Институт Системного Программирования
им. В.П. Иванникова
Российской Академии наук

ISSN 2079-8156 (Print)
ISSN 2220-6426 (Online)

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 30, выпуск 5

Volume 30, issue 5

Москва 2018

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings

Содержание

Table of Contents

Informatics: Formation of computer software and technologies of software systems[1]

[1,3] *E.M. Lavrischeva <lavr@ispras.ru>*
[1,2,4] *A.K. Petrenko <petrenko@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *Moscow Institute of Physics and Technology (State University) 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[4] *National Research University Higher School of Economics (HSE) 11 Myasnitskaya Ulitsa, Moscow, 101000, Russia*

Abstract. Formation of Informatics and aspects of computer software development, in particular, operating systems and information systems since the period of appearance of the first computers of 1948-1990 is considered. The program of informatization of Russia in 1992 and the development of the fundamentals of modern computer science or informatics and the intellectualization of the development of various types of software systems.

Keywords: computer science; computer; system and software engineering; technology; science; information; intelligent systems; digital economy.

DOI: 10.15514/ISPRAS-2018-30(5)-1

For citation: Lavrishcheva E. M., Petrenko A. K. Informatics: Formation of computer software and technologies of software systems. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 7-30. DOI: 10.15514/ISPRAS-2018-30(5)-1

1. Introduction

The emergence of informatics as an independent scientific and engineering discipline was preceded by a period of a large number of theoretical and practical works, the most important of which include the works by C. Babbage, G. Boole, K. Zuse, J. von Neumann, N. Wiener, A. Turing, and others.

The foundation for the development of informatics became research on cybernetics. N. Wiener in his book «Cybernetics: Or Control and Communication in the Animal

[1] This work is supported by RFBR grant 16-00-00352

and the Machine» (1948) defined the term Cybernetics as the science of information, control and communication in animal and inanimate systems. Computer is a tool of this science, control and information processing processes should be based on information about the living nature, be performed using arithmetic operations on data, reflect human mental activity by means of a natural language.

The groundwork for informatics in the USSR was laid in the works by scientists and engineers of the first half of the last century. Historians of science and technology note the significant contribution of such scientists as A.A. Lyapunov, A.P. Ershov, V.M. Glushkov and many others. In the 1950s, the English terms «computer science» appeared. The «software engineering» term appeared in 1968.

Actually, the «informatics» term became widely known in European countries after the appearance in 1973 of the book by F. Bauer and G. Gooz (in Russian translation «Informatics. Introductory Course», Mir, M., 1976).

In the aforementioned book, F. Bauer [1] for the first line of the preface called computer science as informatics, which includes communication technology to determine physiology, psychology, and neurobiology. Informatics touches coding theory, information theory, logical calculus, automata theory, algorithms, algorithmic and intelligent systems. Many of these theories and concepts were formed during the creation of the first computers and promoted the emergence of information systems and technologies, neural systems, nanosystems, etc. In the period when first computers appeared, the ideas of intelligent, smart machines were also formed, solving mathematical problems of algebraic formulation in a special language called «Analyst» on the Mir 1-3 computer (the 1960s) [2], neurocomputers of N.M. Amosov and robots.

A lot of research in the USSR contributed the formation of informatics and cybernetics. The work by D.A. Pospelov «The Formation of Informatics in Russia» (http://www.raai.org/about/persons/pospelov/pages.htm), as well as the USSR conferences on the subject held by him for many years, can be named as particularly publications and in the book of D. A. Pospelov and Y. I. Fet «Essays on the history of Informatics in Russia». IIS SB RAS, 1998, etc. Cybernetics and Informatics, software engineering technologies for newly created computers occupies an important place. Brief description of the types of domestic programming technologies and scientific foundations of informatics are given below.

2. Programming technologies in the USSR in the 1950s and early 1960s

2.1. Programming systems and libraries of programs

Initially, programs for solving various tasks for computers were coded using simple languages such as AutoCode close to a machine language. The first programming program (PP) for A.A. Lyapunov language of operator calculus was developed in 1953-1954 at Moscow State University (E.Z. Lyubimsky, A.P. Ershov).

The library method (V.M. Glushkov «On a Method of Programming Automation», Cybernetics Problems, Issue 2, 1957) is created for solving systems of differential equations for computers (MESM, Ural 1, UMShN, Dnepr, etc.).

Reusable subprograms were collected in subprogram libraries (V.M. Kurochkin, Computing Center of AS of USSR). M.R. Shura-Bura and E.A. Zhogolev developed the technological approach to the management of program libraries in the works. One of the outstanding results in this direction was the so-called «Interpretive System» IS-2 (M.R. Shura-Bura) for the computer Strela.

By the early 1960s, programming languages Algol-60, Fortran, PL/1, Cobol, etc. appeared. Operating systems and translators from the Algol-60 (TA) language were developed for all domestic computers to work with programs in programming languages [5–9].

TA1– S.S. Lavrov (1962).

- TA2 – M.R. Shura-Bura and E.Z. Lyubimsky (IAM, 1963).
- TA3 – (Alpha-system) Russian version of the Algol-60 language, A.P. Ershov (Siberian Branch of the USSR Academy of Sciences, 1964).
- TA4 – E.L. Yushchenko, E.M. Lavrischeva for Dnepr-2 controlled computing complex (Institute of Cybernetics of the Academy of Sciences of the Ukrainian SSR).
- Translator with ALMO (E.Z. Lyubimsky, 1965).

The development of translators built a strong basis for research in the compiler technologies, which are actively developing in our country at the present time.

2.2. Operating systems for BESM-6

The first operating systems in the USSR were created for the BESM computer under the supervision of S.A. Lebedev and M.V. Keldysh as well as at the Joint Institute for Nuclear Research (N.N. Govorun) in Dubna, starting from the 1960s [3, 4].

1) OS-68 or Dispatcher 68 (L.N. Korolev, V.P. Ivannikov, A.N. Tomilin, and M.G. Tchaikovsky). OS provided with:

- Multi-program problem solving;
- Management of all communication channels and devices;
- Parallel execution of tasks and device operation;
- Dynamic allocation of resources in RAM and external memory;
- Distribution and buffering of input-output;
- Operator and interactive mode control. Dispatcher 68 (ND-70) became the basis of the BESM-6 OS, which has been used in the flight control centers of spacecraft, ballistic and telemetric software systems for more than 20 years.

2) IAM OS for BESM-6 began to operate in 1970 (S.S. Kamynin, E.Z. Lyubimsky, and I.B. Zadykhaylo). In this OS, resource management is implemented with the

means of process synchronization and emergency management, and a system of programming and debugging programs in programming languages is included in it.

3) Monitor – Dubna (N.N. Govorun and V.P. Shirikov) ensured management of tasks and library programs creation, including the European programs of CERN and the broadcasting of programs in Fortran, Algol, Pascal, and other programming languages.

The operating systems for BESM-6 supported smart interrupt handling systems, memory protection, real-time and time-sharing modes.

It should be additionally noted that in the mid-1960s (a little bit earlier then OS-68) M.R Shura-Bura and V.S. Shtarkman (Institute of Applied Mathematics - IAM AS USSR) developed the operating system for the Vesna computer, but due to the security reason first and very brief information about this work was published much later.

3. Programming technologies in the early 1970s and 1980s

3.1. Operating systems in 1980-s

A further development of the research in operating systems became OS for multi-machine complex AS-6, in which various types of computers were combined. AS-6 OS design used experience gained in ND-70 development and exploitation. Work on the AS-6 OS has determined the direction of development of the OS and programming systems with OOP for the coming years.

3.2. Compilers in 1980-s

Translator from ALGOL-68 made at Leningrad State University (G.S. Tseitin, A.N. Terekhov, 1968-1991) for the first computers – CM1, CM2, ES EVM, and Sampson (HLL soviet computer). Programs in ALGOL-68 were translated into an intermediate language, and then into a specific machine code.

3.3. Modular and assembly programming

On first domestic computers, programs were initially implemented in machine codes or languages like AutoCode. Later, the programming languages appeared (Algol, Fortran, PL/1, Cobol, etc.). The above-mentioned translators from high-level programming languages had been used on domestic computers and had provided solutions to various kinds of problem domains (mathematical, physical, economic, etc.) before the ES EVM (clones of IBM's System/360 and System/370) appeared.

Within the defense industry specific computers for radars, spaceships, airborne equipment and software complexes PROTVA, YAUZA, RUZA, PROMETHEUS (Moscow Research Institute of Instrument Automatics, MNIIPA, V.V. Lipaev) were created. These complicated systems had a modular architecture, were written in different programming languages and were assembled into complexes using interfaces of the APROP system of the PROMETEY project (1980-1990) [11-13].

The interface has become the main element of the program assembly and the foundation of assembly programming. Its development was influenced by

V.M. Glushkov, V.V. Lipaev, A.P. Ershov, E.H. Tyugu, and others. The APROP system became a CASE-tool for automating the assembly process using primitive functions of the interface library for converting inconsistent data types in modules in the programming language (Lavrischeva E.M., Grishchenko V.N. Linking Multilingual Modules in the Unified Computers Operating System. M.: Finance and Statistics, 1982. 127 p.). The development of this system and its use have been funded by the USSR Ministry of Radio Industry for more than 10 years, being an integral part of the PROMETHEUS, YAUZA, RUZA, and other systems. The APROP systems were transferred to Yerevan Research and Development Center (1984) for using in other Software systems. The main developers of this complex were given the Award of the USSR Cabinet of Ministers (1985). This system was transferred in 52 organizations of the USSR and became an integral part of the ES EVM programming systems. Academician A.P. Ershov considered "assembly programming effective, since ready-made modules make it possible to quickly solve any tasks from a specific problem area for the ES EVM and mini-, micro- and macro computers" (A.P. Ershov. Scientific basis of programming. Report in the USSR Academy of Sciences, 1985, pp. 1-12.).

The results of the research and development in the assembly programming were defended in the Dissertations E.M. Lavrischeva (Models, Methods and Means of Assembly Programming, 1988), Grishchenko V.N. (Methods and Means for Providing the Intermodule Interface in the Class of High Level Programming Languages, 1990) and published in their monograph (Assembling programming, 1991, 136p.) and others articles [25-29]. The implemented interface is built into IBM OS 360 and other foreign operating environments. Later, the ISO/IEC FDIS 24765 – Configuration (assembling), 2009 standard was developed. There was no national standard for the assembly method, and only the domestic assembly concept was published in a number of articles in the Programming and Compute Software Journal (Academy of Sciences of USSR), in books "Technology of assembling programming" by V.V. Lipaev, A.A. Shtrik, B.A. Pozin, published in 1992 and presented in many papers for Scientific conferences in 1976, 1978, 1980, 1986, 1991.

3.4. Programming synthesis

Synthesizing programming (E.H. Tyugu) is the process of obtaining a program from the problem statement and the method of its solution. The synthesis of programs based on logical and analytical specifications and consists in proving the existence theorem and transforming these specifications into a problem-solving program. In case of a logical approach, the specification is interpreted as the formulation of a theorem stating the existence of a solution to the problem. In the case of the analytical approach, the specification is interpreted as an equation for the symbolic transformation of the program with validation. Program synthesis is a certain way of manipulating knowledge embodied in the domain problem specification and universal knowledge that reflects general mathematical patterns and the essence of evidence-based reasoning [13, 17].

3.5. Compositional programming

The composition of programs is the operation of combining functions and data in one of the forms: “data – function – name” and “function-composition-description” on a set of named data, descriptions and denotations (values). Composition operations form a subclass of standard compositions and compositional functions at the programming language level (V.N. Red’ko). The theory of descriptive and declarative programmatic formalisms (N.S. Nikitchenko) provides adequate modeling of data structures, programs, and means of their design. The basis of this theory is semantics, a system of compositions and nominative data for describing functions and compositions. The theory is closely related to function and data algebra and differs from traditional program systems by a theoretical-functional approach, classes of single-valued functions of arity n with nominative data structures such as sets, multisets, relations, etc. An experimental system that displays functions and data compositions, which was used in teaching students of KNU [18-21] and France, had been created.

3.6. Functional programming – REFAL

Functional programming is used to solve problems related to pattern recognition, communication in the natural language, implementation of expert systems, proof of theorems, symbolic calculations, etc. The basis of this programming is the theory of lambda calculus (A. Church) and combinatorial logic (M. Schönfinkel and H. Curry). The functional program includes a set of definitions of functions, calls to other functions and control calls. Recursion is the fundamental basis for constructing semantics in functional programming. In domestic practice, the REFAL language (V. Turchin, 1966) is considered a functional one [22]. A function in it is an ordered set of sentences consisting of a sample and a pattern. Some expression is supplied to the input of the function. It is matched with the sample and if the matching is successful, then this expression will be the result of the function or an error will be recorded. An expression is a sequence of terms that can be letters, numbers, and label characters (identifiers), macro numbers – a character (digital) record of non-negative integers, floating-point numbers, variables from one of several predefined types, and an arbitrary expression in parentheses or in angle brackets to call functions, including recursive ones. This language is still being developed and is practically used in a number of industrial areas [23].

3.7. Layered (aspect oriented) programming

A. L. Fuksman [24] developed the technology of distributed actions or the vertical layering technology. According to this technology, the vertical layer (slice) contains a set of distributed actions, code fragments that implement a certain expanding function, while the process of developing and modifying a program is a sequence of operations for adding or changing these functions. This concept forms the basis for a new strategy for incremental program development. The first stage has a “basis” created – a simplified version of the program. At subsequent stages, the program is

implemented, and it is added to the vertical layer. The program may contain "stubs" – simulators of missing parts for inserting ready-made programs. The transaction is executed as the addition of a new component (vertical layer), which splits into modules designed for several horizontal layers. These modules add security, protection, and reliability. Any program has a series of extensions on the horizontal layer. This programming has received development abroad in aspect-oriented programming [25].

3.8. Algebraic and agent programming (AP)

The AP paradigm is based on the theory of rewriting terms using the equality system of the computation algorithm, the result of which is a term obtained using graph terms of data representation and knowledge of subject domains [26-28]. This device allows determining the behavior of systems and their equivalence in a transitive system. The main state of the transitive system is the system behavior, which is defined by expressions of the behavior algebra F(A) on the set of algebra operations A. Two prefixing operations $a \cdot u$ define the behavior of u on the operation a and the non-deterministic choice $u + v$ of one of two u and v behaviors, which is associative and commutative. The final behavior is given by the constants: Δ, $\perp$, 0 denoting the state of successful, undefined and unsuccessful completion. The behavior algebra is partially defined by the $\leq$ relation, for whom the $\perp$ element is the smallest, and the operations of the behavior algebra are monotone. *The environment E*, where the object is located, is defined as an agent in the algebra of actions *A* and the insertion function of two arguments *Ins (e, u) = e [u]*. The first argument is the behavior of the environment; the second is the behavior of the agent, which is inserted into this environment in a given state. The agent algebra is the parameters of the environment, and the value of the insertion function is a new state of the same environment. AP integrates procedural, functional and logical programming. A term graph is used to represent data and knowledge using expressions of polybasic data (group, ring, and field) algebra. Insertion (insert) programming is formed on the basis of the AP. The AP was implemented on MIR 1-3 computer, and the agent theory was developed for management decisions making by E. A. Trakhtengerts [29].

3.9. Graph programming

The theory of graphs began to develop relatively to programming in the school of A.P. Ershov (V.I. Kasyanov, V.E. Itkin, A.A. Evstigneev and others). The modular programming paradigm is based on the representation of modules and interfaces in graph nodes [11]. The graph is defined as an adjacency matrix and a characteristic vector. The adjacency matrix is used to prove the reachability of graph nodes and the correctness of the modular structure, as well as the formal conversion of the data types transferred between inconsistent modules (see Chapter 3, 4 in the Assembly Programming Book, 1991). Last year Graph theory is being actively developed in

ISP RAS for the schematic organization of memory for Linux and is used in medicine, aviation, biology, etc. The direction of this theory presented in the works [30-31].

3.10. Technology of programming and conferences in the USSR

The definition of programming technology in the USSR was given in the following works:

1. Glushkov V.M. Basic Research and Programming Technology. Programming. 1980, No. 2. p. 3-13.
2. Velbitsky I.V. Programming Technology. K.: Technika, 1984. 277 p.

After the death of V.M. Glushkov (1982), I.V. Velbitsky became responsible for holding the All-Union Conferences of the All-Union Communist Party I-III (1982, 1985, 1992) on behalf of the Executive Committee of the Academy of Sciences of the Ukrainian SSR and the State Committee on Science and Technology of the USSR [33-35]. These conferences gathered thousands of specialists from the country. The following scientists presented scientific reports on programming technologies: A.P. Ershov, V.V. Lipaev, S.S. Lavrov, M.R. Shura-Bura, E.L. Yushchenko and others scientists. The conference materials are stored in the computer science museums (http://www.computer-muse.ru).

4. Informatics and its role in society

N. Wiener (1948) his book on cybernetics in 1948, in discussion on nature of information said "the ideal computing machine must then have all its data inserted at the beginning, and must be as free as possible from human interference to the very end. This means that not only must the numerical data be inserted at the beginning, but also all the rules for combining them, in the form of instructions covering every situation which may arise in the course of the computation. Thus the computing machine must be a logical machine as well as an arithmetic machine and must combine contingencies in accordance with a systematic algorithm».

According to A.P. Ershov, “information is a body of knowledge about actual data, dependencies between them for the cognition and reproduction of human thinking in an artificial intelligence system. Informatics is a science that studies the structure and general properties of information, the laws of the processes of exchange, processing, storage, search and distribution of scientific, financial, economic and other information. Informatics provides mechanisms for reforming documents in natural languages and creating information retrieval, information logical, intelligent systems using system analysis and artificial intelligence. In other words, informatics is considered as a complex scientific discipline, which includes the theory of designing and operation of complex computer, information and intelligent systems (knowledge and data) as well as programming technologies. The term “computer science” began to be used in Russia in the late 1970s, particularly due to the theory of artificial intelligence, which allows simulating human mental activity in the

works of domestic scientists – A.Dorodnitsyn, E.Velikhov, G.Pospelov and D.Pospelov, V.Glushkov, and others [39-44].

4.1. Engineering sciences of computer science teaching

A computer science education program was published in 2005 [R. Shackelford, J.H. Cross, G. Davies, et al. Series "training programs in Informatics", Commission of training programs in Informatics]. It presented the disciplines of Computer Science (Computer Engineering, Electronic Engineering, Software Engineering, System Engineering, IT – Information Technology, Information Systems). A computer science-training program for each individual discipline is provided at https://curriculum.code.org. Below, their brief definitions are provided (Fig. 1).

Information technologies are technologies for managing solutions, business, commerce, economics, social sphere, etc.).

Information Systems – information processing systems (exchange of information, documents, data, etc.).

Software Engineering – methods, means and tools for the development, operation, maintenance and termination of programs usage. It occupies a central place in computer science.

System Engineering – the theory, methods, and principles of software design for computers (OS, translators, interpreters, analyzers, schedulers, etc.), information and software systems maintenance, data management, etc.

Computer Engineering – the theory and principles of building computers, supercomputers, multiprocessor and macroconveyor machines, clusters, etc. The basis of Computer Engineering includes Turing machine, von Neumann, automata, algorithms, mathematics, logics, and other theories.

4.2. Intellectual systems

The intellectual system performs creative tasks, the knowledge of which is stored in its memory. Such a system includes a knowledge base, a decision output mechanism, and an intelligent interface. Intellectual systems are studied by means and methods of artificial intelligence [49].

The main tasks of artificial intelligence:

- symbolic modeling of thinking processes,
- working with natural languages,
- presenting and using knowledge
- machine learning,
- biological modeling of artificial intelligence,
- robotechnics.

These systems were on the agenda as soon as computers appeared. A number of scientists have been studying these problems to the present day [50]. During the period of 1985-1990, specialists from Japan were engaged in the issues of the

intellectualization of knowledge and the construction of smart machines. First Japanese robots became the result of this project [49].

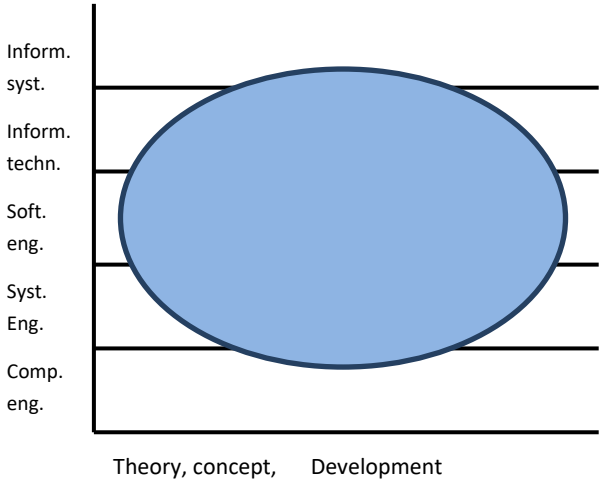

Fig. 1. Informatics space (Software Engineering in the middle)

Information systems and information technologies in computer science have become the main tool for the everyday life and interactions of modern society and the basis for computerization and informational support of the global world.

Today, in a time of the digital economy, the solution of machine learning problems, the use of accumulated knowledge, machine learning and robotization are on the agenda of the government and the Ministry of Education of the Russian Federation. A great contribution to the development of artificial intelligence was made by G.S. Pospelov, who has been holding thematic conferences on Artificial Intelligence in Russia for 10 years.

4.3. Knowledge engineering in engineering domains

Knowledge of a certain subject area (biology, chemistry, physics, etc.) is presented in the Knowledge Base (KB), which contains inference rules and information on human experience and knowledge with consideration of the ISO/IEC 2382-1: 19931-BR standard. Not only is the actual information of the region placed in the KB, but also the rules of inference for performing automatic deduction about the existing or newly entered facts and thereby is semantic (meaningful) processing of information carried out. Modern KBs work in conjunction with the search and retrieval systems for information represented by ontology as a set of concepts and their relationships. Ontologies are described using special ontology languages (OWL, FODA, ODM, etc.) [51, 52]. One of the authors, together with the students of MIPT, developed the ontology of the life cycle domain in OWL, which contains information about processes and actions, as well as artifacts and dependencies between them. The life cycle description in OWL was processed by Protégé 2.3 and

the output code in XML was obtained [53]. The life cycle automation approach was reported at the Science and Information-2015 conference in London. Today, the ontological approach is gaining momentum on the Internet, widely represented in the Semantics Web (http://semanticweb.org) for building ontologies, knowledge extraction (Data Mining), neural, quantum and intelligent technologies.

5. The era of informatization and market oriented economy (1992-2018)

After the collapse of the USSR, the «Informational Support of Russia» Scientific and Technical Council (1992) was established to determine the focus area on the development of information technology for the upcoming years. The council included 30 leading specialists from institutes of the Russian Academy of Sciences and other research organizations of Russia. This council has developed an informational support strategy for Russia, consisting in the development of concepts and principles for the development of promising and typical informational support projects, unification and standardization of design solutions based on international standards and information and communication technologies (ICTs) that improve the efficiency of informational support, security, quality of information technologies and economic mechanisms of informational support [62].

The basics of informational support of Russia included the following areas:

1. Economic and social tasks of informational support.
2. Technologic and economic development of the informational support market economy.
3. Scientific and technical informational support of Russia:
    3.1. Fundamental development of IT and systems.
    3.2. Technology and software engineering systems development.
    3.3. Ensuring information protection.
    3.4. Standardization of system creation tools.
    3.5. Support for informational support in education.
    3.6. Development of informational support infrastructure in Russia.
4. Development of informational support infrastructure in Russia.
5. System projects for informational support in Russia.

The main focus of the informational support project was on the justification of a market economy for various informational support projects.

In area 3.1, the following are related to the scientific areas of informational support (V.P. Ivannikov):

- algebraic and logical theories of system formalization;
- analysis of parallel and distributed systems;
- formal specifications and verification of complex systems;
- programming paradigms;

- knowledge presentation formalisms;
- models of human-machine interfaces and visualization.

The area 3.2 defines the main tasks of software engineering (V.V. Lipaev): development methods, safety, reliability, protection, quality and standardization of the life cycle of information systems, as well as a strategy for developing the market economy of software and hardware and organization of education.

The infrastructure and informational support program of Russia included: a communication system for computing tools and networks to provide interaction of information facilities and technologies; programs for the hardware, media and software complexes; a personnel education system for the effective development of the country's informational support. The market economy is defined as the main tool of informational support.

5.1. Directions of technology development after 1992

The presented domestic programming technologies have come a long way of development and have played an important role in the development of programming of software and system complexes for various purposes on a computer. Practically developed domestic technologies were not brought to the standard in this country, but many standards appeared in software and computer engineering abroad after 1992 [14].

These include, for example: UML (Unified Modeling Language), OWL (Ontology Web Language), WSDL (Web Service Definition Language), XML, XSL, RDF, and others. Since 2002, the Model (Model Variability) has been formed on programs production Product Lines/Product Family [32] and in Grid systems. It is used in AppFab software factories [33].

In domestic practice, a theory of object-component modeling of software systems [36] and a data type conversion system have been developed, taking the ISO/IEC 11404 GDT standard and Big Data non-structural data types into account [Lavrischeva E.M., Ryzhov A.G. Application of the theory of general data types of ISO / IEC 12207 GDT standard to Big Data].

5.2. Industrial technologies after 2005

A few words are to be said on one important direction in the development of computer science, which gained momentum in the early 2000s, related to such concepts as program families and program factories. A program factory is an integrated architecture of pipeline assembly and fabrication of Programming Products (PP) from ready-made resources, which are standardized in programming languages, and their interfaces – standardized in the WSDL language [56-59, 64].

The program factory includes the following elements:

- ready-made resources (artifacts, modules, programs, systems, reuses, assets, objects, components, services, etc.);
- interfaces in one of the languages IDL, SIDL, WSDL, etc.;
- technological and product lines;

- assembly line;
- production and planning techniques;
- configuration of custom product versions;
- development of an environment for individual elements of ready-made resources and functions.

Creation of a product from ready-made elements depends on their volume or quantity. The level of programming programs production using the following functions:

$v = F(z, u)$, where $v = (v_i)$ is the output vector, $z = (z_i)$ is the resource expenditure vector, $u = (u_i)$ is the expenditure function dependency matrix $z = F(w_j, u)$, $j = 1, 2 .., n$. The indicators of the w_j function set the volume of production, the structure of the production assets and the level of ready-made resources specialization of the factory.

The overall Cobb-Douglas production function of output has the form $v = ne^{\pi} L^{\alpha} K^{\beta}$, where v is the generalized output, n is the normative factor, e is the basis of the algorithm, t is an indicator of the level of scientific and technical progress, L is the cost of human labor, K – the amount of capital, α, β – elasticity coefficients.

The calculations of these functions determine the estimates of the revenue part of the production at the program factory. The efficiency, consistency, and proportionality of PP output depend on resource management by factory services (control, testing, quality assessment of PP, etc.). In a market economy, the quality and cost of PP evaluations correspond to the demand for a product.

The following factories are known for PP production at the moment:

- Product Line/Product Family (K. Pohl) [32].
- Conveyor multigeneration (K. Czarnecki) [48].
- AppFab in VS.Net (https://msdn.microsoft.com/ru-ru/library/ee677312(v=azure.10).aspx).
- IBM Sphere (https://www.ibm.com/developerworks/ru/websphere/newto/).
- Streaming assembly by J. Greenfield [58].
- Chart assembly use case by G. Lenz and M. Fowler's continuous integration.
- Programs assembling according to V. Glushkov [47, 64].
- BEA WebLogic Oracle, SAP NetWeaver, and other factory systems.

For planning and making economic calculations at the factories programs was development special disciplines [46-48]. The production of *PP* is based on the technological processes of manufacture of certain product types using the theory of the design and usage of tool environments. These disciplines are used in the program CS-Curricula-2013 and was demanded many organizations of China, India and other countries [39, 45, 54].

Brief description of Software Engineering (SE) domains is presented below.

Scientific disciplines of SE include classic Sciences (theory of algorithms, set theory, proof theory, mathematical logic, discrete mathematics); theory of

programming of the theory of abstract data, management science, etc. This discipline defines the basic concepts of the objects and the formalism of the description of the system components and data description [15-20] etc.

Engineering disciplines of SE include methods of using technology rules and procedures, processes, life cycle, methods of measuring and assessing the quality of development *PP*. This discipline defines the set of engineering methods, techniques, tools and standards focused on the production of the target PP. Basic concepts of engineering SE include: core knowledge SWEBOK; the basic processes of SE; infrastructure environment [45].

Project management is based on the theory of management of projects and the IEEE Std.1490 PMBOK (Project Management Body of Knowledge); method CRM (Critical Path Method) for the graphic representation of works, operations and their execution time; method of network planning PERT (Program Evaluation and Review Technique), etc. In the PMBOK defined processes lifecycle of the project and the main areas of knowledge and processes of planning, monitoring, management and completion.

Economic disciplines. This disciplines are used for the analysis of the different parties activities of developers in the implementation of the project and identify the costs. The time and economic indicators are according to the requirements of *PP*. Used methods: predicting the size of *PP* (FPA – Function Points Analyses, Feature Points, Mark II Function Points, 3D Function Points, etc.); the evaluation effort for the development of *PP* by using models COCOMO and systems (Angel, Slim, Seer-SEM, etc.), as well as the quality of *PP* [35].

Production disciplines determine the production of *PP*. In software industry mass software products are produced by such worldwide leaders as Microsoft, IBM, Intel, Oracle, and other factory programs owners [13, 58, 59, 64].

5.3. The way of automation systems modeling

This paper does not assume a detailed analysis of the current state of computer science neither in Russia nor abroad. Nevertheless, it can be stated that intellectual systems are nowadays the main direction of computer science development. An important area of research in this direction is the development of robots in the manufacturing industry, medicine and other areas of their application. Robot automation is understood as the automation of the production of individual objects based on the usage of industrial unmanned technologies. The purpose of robot automation is to improve working conditions, increase technical and economic indicators of enterprises, and ensure the highest and most efficient production. This area is the most developed in Japan. They were one of the first to develop robots, starting from the late 1960s – early 1970s. Today, this country plans to launch the first unmanned taxi. In 2015, a prototype was presented bearing the name of Robot Taxi. It was shown in 2016 at the top level – during the G7 summit. It is expected that the full implementation of such machines in the market will occur in 2020 and be prepared by the time of the Olympics in Tokyo.

In terms of the evolution of design technologies, the development and analysis programs, the problems of ensuring information security, reliability and resiliency of software and hardware of the critical infrastructure come to the fore. One of the important aspects of the reliability and information security of software systems is designing and verification of so-called "variable" software systems or software product families.

One of the important aspects of reliability and information security of computer systems is modeling and verification of «variable» systems or software product families. The project of RFBR 16-01-00352 «Theory and methods of development of variable software systems» (head E. M. Lavrisheva) is devoted to the solution of a number of problems of modeling of variable systems. These include the implementation of the following tasks:

- modeling of software, operating and Web systems from ready-made software resources (GOR) by OCM method and creation of *MF* (Feature Model), *Msys* and *Mconfig* models [58-63];
- verification of data models, GOP, information and Web systems from using of services and services component, which founding in Internet libraries reuses and CPI [67-74] ;
- testing of GOR and configured systems on models *Mconfig*, *MF*, *Msys* with data collection about errors, defects, failures, etc. [54];
- integrated testing of evaluation of reliability and quality of the created the information and Web systems [74].

A description of the methods to implement the above processes, modeling of complex software, information and web-systems are given in the works of the participants of the project RFBR 16-00-00352. The main results of this project are published in [65-74].

6. Conclusions

The ways of establishing informatics and programming technologies for automating the creation of software and application systems since the advent of the first Electronic computers, as well as the new discipline for processing information on both the first computers and modern ones during the period of the informational support, are considered. Computer and information systems, as well as technologies for the automation and intellectualization of development of software and application systems of various types, are provided (the program of informational support of Russia after 1992 is described. According to the tasks developed and described new domestic technologies and tools of systems analysis, modeling of system and site with the help of ready GOR (objects, components, services, etc.) on new variable models and method of Configuration (assembling) are presented. There are new ways of development of intelligent systems in the direction of robotics devices, machines, etc. for different areas, especially for medicine. Innovative developments in the field of intellectualization, ontologization and ensuring the reliability and information security of software systems are noted.

References

[1] Bauer F.E., Gooz G. Informatics. M: 1976, 484 p. (in Russian)
[2] V. M. Glushkov "Fundamentals of paperless Informatics" // M.: Nauka, 1982, 552 p.
[3] Ivannikov V.P., Korolev L.N., Lyubimsky E.Z. et al. Development of the Moscow school of computer OS. International Symposium Computers in Europe: Past, present and future, Kyiv, Ukraine, 5-9 October 1998, pp. 265-270 (in Russian)
[4] Ivannikov V.P., Gontarenko S.V., Govorun N.N. Architecture of operating systems for supercomputers. Problems of Cybernetics. Problems of creating high-performance computers, 1984. pp. 117-126 (in Russian).
[5] Shura-Bura M.R., Lyubimsky E.Z. Translator from ALGOL-60. Computational Mathematics and Mathematical Physics, vol. 4, № 1, 1964 (in Russian).
[6] Babetskii G.I., Bezhanova M.M., Ershov A.P. et al. Programming System ALPHA.- Computational Mathematics and Mathematical Physics, №2, 1965, pp. 317-325 (in Russian).
[7] Lavrischeva E.M., Borisenko, L.G., Grishkevich E.I. et al. The translator from the language of D-ALGAMS for UVK Dnepr-2. K: IK Ukrainian Academy of Sciences, 1970. - 186 p. (in Russian)
[8] Terekhov A.N. Identification and structure of a compiler language ALGO-68. Programmirovanie, N2, 1975 (in Russian)
[9] A.P. Ershov and M.R. Shura-Bura. The Early Development of Programming in the USSR. In History of Computing in the Twentieth Century, Academic Press; First Edition edition, October 12, 1980, pp. 125-136. (in Russian)
[10] Safonov V.O. Languages and methods of computer programming Elbrus. M.: Nauka, 1989
[11] E.M. Lavrishcheva, V.N. Grishchenko. Assembly programming. K.: 1991, 213 p. (in Russian)
[12] V.V. Lipaev, B.A. Posin, A.A. Shtrik. Technology of Assembly programming. M.: 1992, 271 p. (in Russian)
[13] E.M. Lavrishcheva, V. N. Grishchenko. Assembly programming. The basics of the industry systems", Kiev, Naukova dumka, 2009.-371 p. (in Russian)
[14] Lipaev V.V. Processes and standards of life cycle of complex software tools. Reference book. M.: SYNTEG, 2006, 260 p. (in Russian)
[15] Lavrishcheva E.M. Development of the theory of programs and systems in the USSR: History and modern theories. In Proc. of the Fourth International Conference on Computer Technology in Russia and in the Former Soviet Union (SORUCOM). Moscow, Zelenograd, 2017, pp. 31-43.
[16] Kahro M.I., Kalia, A.P., Tyugu E.X. Instrumental programming system UCS (PRIZ). M.: Finance and statistics, 1981 (in Russian).
[17] Tyugu E.X. Conceptual design. M., Nauka: 1984, 287 p. (in Russian)
[18] Red'ko V.N. Program compositions and composition programming // Programmirovanie, № 5, 1978, pp. 17-26. (in Russian)
[19] Nikitchenko N.S., Shkilnyak S. S. Mathematical logic and theory of algorithms. K.: Kiev University: 2008, 528 p. (in Ukrainian)
[20] Lavrischeva E.M., Nikitchenko N. S. Omelchuk L.P. Programming technology of information systems: methods, tools, tools. K., Kiev University: 2017, 457p.
[21] Systems of computer algebra of the ANALYTIC family, group of authors. K.: NASU, IPMI, 2010, 762 p. (in Russian)

[22] Turchin V. F. the Algorithmic language of recursive functions (REFAL). Moscow: Preprint of IAM AN SSSR, 1968 (in Russian)
[23] Smirnov V. K., the Hardware realization of the language Refal in IPM im.M. V. Keldysh. Preprint KIAM RAS, №99, 2003, pp. 1-21 (in Russian).
[24] Fuksman, A. L. Technological aspects of building software systems. M.: Statistika, 1979, 183 p. (in Russian).
[25] Aspect-oriented programming. AspectJ (http://aspect.org).
[26] Kapitonova Y.V., Letichevsky A.A. Methods and Means of Algebraic Programming. Kibernetika/Cybernetics, 1993, issue 3, pp. 7-12 (in Russian).
[27] Letichevsky A., Gilbert D. A general theory of action languages. Cybernetics and Systems Analysis, vol. 34, no. 1, 1998, pp. 12-30 . DOI: 10.1007/BF02911258.
[28] Trakhtengerts E. A. The Interaction of agents in multi-agent environments. Automation and telemechanics, № 8, 1998, pp. 3-52 (in Russian).
[29] Trachtengerts E. A. Computer methods of realization of economic and information management decisions. Volume 1. Methods and means. Volume 2 the Implementation of solutions. Moscow, 2009 (in Russian).
[30] A.A.Evstigneev. Application of graph theory in programming. Moscow, Nauka, edited by A.P. Ershov, 1985, 351 p. (in Russian).
[31] Christofides N. Graph theory. Algorithmic approach. M.: Mir. 1978 (in Russian)
[32] Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.
[33] Lavrischeva E.M. Theory and Practice of Software Factories. Cybernetic and Systems Analyses, vol.47, No.6, 2011, pp.961-972.
[34] J.Hebeler, M.Fisher, R.Blace, A.Perez-Lopez. Semantic Web Programming. Wiley Publiching.Inc., 2008, 565 p.
[35] Lavrishcheva E.M. Software engineering and programming technology of complex systems. Moscow, Yurayt, 2018, 432 p. (in Russian).
[36] Lavrishcheva E.M. The theory of object-component modeling of modified software systems. Preprint of ISP RAS, 2016. ISBN 998-5-91474-025-9 (in Russian),
[37] E.M. Lavrischeva, A.K. Petrenko. Software Product Lines Modeling. Trudy ISP RAN/Proc. ISP RAS, vol, 28, issue 6, 2016, pp. 49-65 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-4
[38] Catalogue of Technologies. ISP RAS. Moscow, 2017, 34 p. http://www.ispras.ru/downloads/ISPRAS-Catalogue-En.pdf
[39] E.P. Velikhov. Computer science is an important direction of Soviet science. In Cybernetics. The formation of Informatics, 1986 (in Russian).
[40] A.A. Dorodnitsin. Informatics. Subject and tasks In Cybernetics. The formation of Informatics, 1986 (in Russian).
[41] V.S. Mikhalevich et al. Informatics – new field of science and practice. In Cybernetics. The formation of Informatics, 1986 (in Russian).
[42] Shileiko A., Shileiko T. Conversations about informatics. Moscow, Molodaya Gvardia, 1989, 287 p. (in Russian)
[43] V.M. Glushkov. Cybernetics, computers, informatics. K.: Naukova Dumka, 1990 (in Russian).
[44] V.M. Glushkov. The Basics of Paperless Informatics. M.: Nauka, 1982, 552 p. (in Russian).

[45] Curricula Recommendations. Available at: https://www.acm.org/education/curricula-recommendations
[46] E.M. Lavrischeva. Classification of software engineering disciplines. Cybernetics and Systems Analysis, vol. 44, Issue 6, 2008, pp. 791–796.
[47] Lavrishcheva E.M. Software engineering. Training manual in 3 parts. Moscow: MIPT, 2016 (in Russian).
[48] Czarnecki K., Eisenecker U. Generative programming. Methods, tools, application. SPb.: Peter, 2005, 730 p. (in Russian).
[49] H. Ueno, V. Isudzuka. Representation and use of knowledge. M.: Mir, 1987, 220 p. (in Russian).
[50] G.S. Pospelov. Artificial intelligence – new information technology. In Cybernetics. The formation of Informatics, 1986 (in Russian).
[51] Standard ISO / IEC 2382-1:1993, Information technology – Vocabulary – Part 1: Fundamental terms.
[52] T.A. Gavrilova et al. Knowledge bases of intelligent systems. Textbook for technical universities. SPb.: Peter, 2000 (in Russian)
[53] Lavrischeva E.M. Ontology of Domains. Ontological Description Software Engineering Domain – The Standard Life Cycle. Journal of Software Engineering and Applications, vol. 8 No. 7, 2015.
[54] Lipaev V. V. Software engineering of complex custom software products. Tutorial. M.: MAKS-press, 2014 (in Russian)
[55] E.M. Lavrishcheva, L.E. Karpov, A. N. Tomilin. Approaches to the representation of scientific knowledge in the Internet science. In Proc. of the XIX all-Russian scientific conference "Scientific service on the Internet", 2017, pp. 310-326 (in Russian).
[56] Lavrischeva Ekaterina. (2015). Ontological approach to the formal specification of the standard life cycle. In Proc. of the 2015 Science and Information Conference (SAI), 2015, pp. 965-972.
[57] Lavrishcheva E.M. Fundamentals of software engineering. In Proc. of the 5th International conference on Actual problems of system and software engineering. CEUR Workshop Proceedings, vol. 1989, 2017, pp. 163-177 (in Russian).
[58] Lavrischeva K.M. Theory and Practice of Software Factories. Cybernetic and Systems Analyses, vol. 47, no. 6, 2011, pp. 961-972.
[59] Lavrischeva K., Aronov A., Dzyubenko A. Programs Factory – A conception of Knowledge Representation of Scientific Artifacts From Standpoint of Software Engineering. Computer and Information Science, Vol. 6, No. 3, 2013, pp. 21–28.
[60] Gorodnyaya L.V. Programming Paradigms: Analysis and comparison. Novosibirsk, SB RAS, 2017, 223 p. (in Russian).
[61] Lavrischeva E.M. Software Engineering. Programming technology. Moscow, MIPT, 2016, 52 p. (in Russian).
[62] Scientific and technical bases of informatization of Russia. Ministry of Science and Technical Policy of the Russian Federation. Moscow, 1992, 151 p. (in Russian).
[63] E.M. Lavrischeva, I.B. Petrov, Ways of Development of Computer Technologies to Perspective Nano. In Proc. of the Future Technologies Conference (FTC-2017), 2017, pp.539-547.
[64] Lavrischeva E. M. Assembly line of program factories – the idea of academician V. M. Glushkov. In V. M. Glushkov: The past is looking to the future. K.: Academperiodica, 2013, p. 130-143 (in Ukrainian)

[65] E.M. Lavrischeva, L.E. Karpov, A.N. Tomilin. Semantic resources for the development of ontology of scientific and engineering subject areas, In Proc. of the XVIII all-Russian scientific conference "Scientific service on the Internet", 2016, pp. 223-239 (in Russian).

[66] Lavrishcheva E.M. Theoretical foundations of modeling software systems from objects and components. In Proc. of the International scientific-practical conference on Theory of active systems (TAS-2016), 2016, pp. 314-325 (in Russian).

[67] Kuliamin. V.V., Lavrishcheva E.M., Mutilin V.S., Petrenko A.K. Verification and analysis of variable operating systems. Trudy ISP RAN/Proc. of ISP RAS, vol. 28, issue 6, pp. 48-59 (in Russian). DOI: 10.15514/ISPRAS-2016-28(3)-12

[68] Lavrischeva E. Assembling Paradigms of Programming in Software Engeneering. Journal of Software Engineering and Applications, vol. 9, no. 6, 2016, pp. 296-317.

[69] Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle. In Proc. of the Science and Information Conference, 2015, pp. 965-972.

[70] Lavrischeva E. M., V. S. Mutilin, A. G. Ryzhov. Aspects of modeling of variable software and operating systems. In Proc. of the XIX all-Russian scientific conference "Scientific service on the Internet", 2017, pp 327-341 (in Russian).

[71] Lavrischeva E.M., Mutilin V.S., Ryzhov A.G. Designing variability models for software, operating systems and their families. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 93-110. DOI: 10.15514/ISPRAS- 2017(5).

[72] Lavrischeva E. Scientific Basis of System Programming. Journal of Software Engineering and Applications, No. 11, pp. 408-434.

[73] E.M. Lavrischeva. The Scientific basis of Software Engineering. International Journal of Applied and Natural Sciences, vol. 7, issue 5, 2018, pp. 15-32.

[74] Lavrischeva E.M., Pakulin N.V., Ryzhov A.G., Zelenov S.V. Analysis methods for assessing the reliability of equipment and systems. Application practice methods'. Trudy ISP RAN/Proc. ISP RAS, vol.30, issue 3, 2018, pp.99-120 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-8

Информатика: становление программного обеспечения и технологий программных систем

[1, 3] *Е.М. Лаврищева <lavr@ispras.ru>*
[1, 2, 4] *А.К.Петренко <petrenko@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский Государственный университет имени М. В. Ломоносова Москва, 119991, ГСП-1, Ленинские горы, д. 1*
[3] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[4] *НИУ "Высшая школа экономики", 101000, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. Рассматривается становление информатики и аспектов компьютерной поддержки создания программного обеспечения, в том числе системных и прикладных

программных систем, начиная от периода появления первых ЭВМ 1948-1990-х годов XX века. Рассмотрена программа информатизации России 1992 года и пути развития фундаментальных основ современных компьютерных наук или информатики и интеллектуализации разработки различных видов программных систем.

Ключевые слова: информатика; компьютерная, системная и программная инженерия; технология; наука; информационные системы; интеллектуальные системы; индустрия; цифровая экономика.

DOI: 10.15514/ISPRAS-2018-30(5)-1

Для цитирования: Лаврищева Е.М., Петренко А.К. Информатика. Становление программирования и технологий программных систем. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 7-30. DOI: 10.15514/ISPRAS-2018-30(5)-1

Список литературы

[1]. Ф. Бауэр, Г. Гооз. Информатика. М.: Мир, 1976, 484 с.
[2]. Глушков В.М. Основы безбумажной информатики. М.: Наука, 1982, 552с.
[3]. Иванников В.П., Королев Л..Н., Любимский Э.З. и др. Разработка Московской школы ОС ЭВМ. Международный симпозиум «Компьютеры в Европе. Прошлое, настоящее и будущее (Computers in Europe. Past, Present and Future) 5-9 октября 1998. Киев, Украина, 265-270 стр.
[4]. Иванников В.П., Гонтаренко С,В., Говорун Н.Н. Архитектура ОС суперЭВМ.- Вопросы кибернетики. Проблемы создания высокопроизводительных ЭВМ.- Совет по проблеме «Кибернетики», 1984. -М., стр.117-126.
[5]. Шура-Бура М.Р., Любимский Э.З. Транслятор с языка Алгол-60. Журнал вычислительной математики и математической физики, том 4, № 1, 1964.
[6]. Бабецкий Г.И., М.М.Бежанова, Ершов А.П. и др. Система программирования АЛЬФА. Журнал вычислительной математики и математической физики, №2, 1965, стр. 317-325.
[7]. Лаврищева Е.М., Борисенко Л.Г., Гришкевич Е.И. и др. Транслятор с языка Д-АЛГАМС для УВК Днепр-2. К.: ИК АН УССР, 1970, 186 стр.
[8]. Терехов А.Н. Проблемы идентификации и структура компилятора с языка АЛГОЛ 68. Программирование, No 2, 1975, стр. 61-67.
[9]. А.П.Ершов, М.Р.Шура-Бура. Становление программирования в СССР (переход ко второму поколению языков и машин). Новосибирск: Изд. ВЦ СО АН СССР, Препринт №13, 1976.
[10]. В.О. Сафонов. Языки и методы программрования для ЭВМ «Эльбрус», М.: Наука, 1989, 389 стр.
[11]. Е.М. Лаврищева, В.Н. Грищенко. Сборочное программирование. Наукова думка, Киев: 1991, 213 стр.
[12]. В.В. Липаев, Б.А. Позин, А.А. Штрик. Технология сборочного программирования. Москва:1992, 271 стр.
[13]. Е.М. Лаврищева, В.Н. Грищенко. Сборочное программирование. Основы индустрии систем, Киев.: 2009, 371 стр.
[14]. Липаев В.В. Процессы и стандарты жизненного цикла сложных программных средств. Справочник. Москва, Синтег. 2011, 260 стр.

[15]. Лаврищева Е.М. Развитие теории программирования в СССР. История и современные теории. Sorucom-17. Развитие ВТ в России и странах бывшего СССР, 2017, стр. 162-177.
[16]. Кахро М. И., Калья А. П., Тыугу Э. Х. Инструментальная система программирования ЕС ЭВМ (ПРИЗ). М.: Финансы и статистика, 1981.
[17]. Тыугу Э. Х. Концептуальное проектирование. М.: Наука: 1984, 287 стр.
[18]. Редько В.Н. Композиции программ и композиционное программирование. Программирование, № 5, 1978, стр. 17–26.
[19]. Никитченко Н.С., Шкильняк С.С. Математическая логика и теория алгоритмов. К., ИПЦ Киевский университет, 2008, 528 стр.
[20]. Лаврищева Е.М., Никитченко Н.С., Омельчук Л.П. Технология программирования информационных систем. К.: ВШН КНУ, 2010, 351 стр.
[21]. Системы компьютерной алгебры семейства АНАЛИТИК, Коллектив авторов. К.: НАНУ ИПММИ, 2010, 762 стр.
[22]. Турчин В. Ф. Алгоритмический язык рекурсивных функций (РЕФАЛ). М.: ИПМ АН СССР, 1968.
[23]. Смирнов В.К. Аппаратная реализация языка Рефал в ИПМ им.М.В.Келдыша. Препринт ИПМ им. М.В.Келдыша, № 99, 2003, 21 стр.
[24]. Фуксман А.Л. Технологические аспекты создания программных систем. М.: Статистика, 1979, 183 стр.
[25]. Аспектно-ориентированное программирование. AspectJ (http://aspect.org).
[26]. Капитонова Ю.В., Летичевский А.А. Методы и средства алгебраического программирования. Кибернетика, № 3, 1993, стр. 7–12.
[27]. Letichevsky A., Gilbert D. A general theory of action languages. Cybernetics and Systems Analysis, vol. 34, no. 1, 1998, pp. 12-30 . DOI: 10.1007/BF02911258.
[28]. Транхтенгерц Э.А. Взаимодействие агентов в многоагентных средах. Автоматика и телемеханика, № 8, 1998, стр.3-52.
[29]. Трахтенгерц Э.А. Компьютерные методы реализации экономических и информационных управленческих решений. Том 1. Методы и средства. Том 2 Реализация решений. М.:2009.
[30]. Евстигнеев А.А. Применение теории графов в программирование. Москва.-Наука, 1985, 351стр.
[31]. Кристофидес Н. Теория графов. Алгоритмический подход. М.: Мир. 1978.
[32]. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.
[33]. Lavrischeva E.M. Theory and Practice of Software Factories. Cybernetic and Systems Analyses, vol.47, No.6, 2011, pp.961-972.
[34]. J.Hebeler, M.Fisher, R.Blace, A.Perez-Lopez. Semantic Web Programming. Wiley Publiching.Inc., 2008, 565 p.
[35]. Лаврищева Е.М. Программная инженерия и технология программирования сложных систем. Москва, Юрайт, 2018. 432 стр.
[36]. Е.М. Лаврищева. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН №29, 2016, 52 стр. ISBN 978-5-91474-025-9
[37]. Лаврищева Е.М., Петренко А.К. Моделирование семейств программных систем. Труды ИСП РАН, том 28, вып. 6, 2016, стр. 49-64. DOI: 10.15514/ISPRAS-2016-28(6)-4
[38]. Каталог технологий ИСП РАН. Москва, 2017, 34стр.

[39]. Е.П. Велихов. Информатика – актуальное направление советской наукию. В сб. "Кибернетика. Становление информатики", М.:Наука, 1986.
[40]. Дородницын А.А. Информатика: предмет и задачи. В сб. "Кибернетика. Становление информатики", М.:Наука, 1986.
[41]. В.С. Михалевич и др. Информатика – новая область науки и практики В сб. "Кибернетика. Становление информатики", М.:Наука, 1986.
[42]. Шилейко А., Шилейко Т. Беседы об информатике.-Москва. Молодая Гвардия, 1989, 287 стр.
[43]. В.М. Глушков. Кибернетика. Вычислительная техника. Информатика. Киев, Наукова думка, 1990..
[44]. В.М. Глушков «Основы безбумажной информатики». М.: Наука, 1982, 552 стр.
[45]. Curricula Recommendations. https://www.acm.org/education/curricula-recommendations.
[46]. E.M. Lavrischeva. Classification of software engineering disciplines. Cybernetics and Systems Analysis, vol. 44, issue 6, 2008, pp. 791–796.
[47]. Лаврищева Е.М. Программная инженерия. Учебно-методическое пособие в 3-х частях. М.: МФТИ, 2016.
[48]. Чернецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение. Издательский дом «Питер», 2005, 730 стр.
[49]. Х. Уэно, Т. Кояма, Т. Окамото, Б. Мацуби, М. Исидзука. Представление и использование знаний. Москва, Мир,1987, 220 стр.
[50]. Г.И. Поспелов. Искусственный интеллект – новая информационная технология В сб. "Кибернетика. Становление информатики", М.:Наука, 1986.
[51]. ISO/IEC 2382-1:1993, Information technology — Vocabulary — Part 1: Fundamental terms.
[52]. Т.А.Гаврилова и др. Базы знаний интеллектуальных систем. Учебник для вузов. СПб.: Питер, 2000.
[53]. Lavrischeva E.M. Ontology of Domains. Ontological Description Software Engineering Domain – The Standard Life Cycle. Journal of Software Engineering and Applications, vol. 8 No. 7, 2015.
[54]. Липаев В.В. Программная инженерия сложных заказных программных продуктов. Учебное пособие. М.: МАКС-ПРЕСС, 2014.
[55]. Е.М. Лаврищева, Л.Е. Карпов, А.Н. Томилин. Подходы к представлению научных знаний в Интернет науке. Труды. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», 2017, стр. 310-326.
[56]. Lavrischeva Ekaterina. (2015). Ontological approach to the formal specification of the standard life cycle. In Proc. of the 2015 Science and Information Conference (SAI), 2015, pp. 965-972.
[57]. Лаврищева Е.М. Фундаментальные основы программной инженерии. Труды 5-международной конференции «Актуальные проблемы системной и программной инженерии, 2017, стр.163-177.
[58]. Lavrischeva K.M. Theory and Practice of Software Factories. Cybernetic and Systems Analyses, vol. 47, no. 6, 2011, pp. 961-972.
[59]. Lavrischeva K., Aronov A., Dzyubenko A. Programs Factory – A conception of Knowledge Representation of Scientific Artifacts From Standpoint of Software Engineering. Computer and Information Science, Vol. 6, No. 3, 2013, pp. 21–28.
[60]. Городняя Л.В. Парадигмы программирования: Анализ и сравнение. Новосибирск: Изд-во СО РАН, 2017 г., 232 с

[61]. Лаврищева Е.М. Программная инженерия. Технология программирования. Москва, МФТИ, 2016.

[62]. Научно-технические основы информатизации России. Мин. Науки, высшей школы и технической политики РФ. Москва, 1992, 151 стр.

[63]. E.M. Lavrischeva, I.B. Petrov. Ways of Development of Computer Technologies to Perspective Nano. In Proc. of the Future Technologies Conference (FTC-2017), 2017, pp. 539-547.

[64]. Лаврищева Е.М. Сборочный конвейер фабрик программ – идея академика В.М.Глушкова. В книге «В.М.Глушков: Прошлое устремленное в будущее». К.: Академпериодика, 2013, стр. 130—13.

[65]. Лаврищева Е.М., Карпов Л.Е., Томилин А.Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей. Труды. XVIII Всероссийский научной конференции «Научный сервис в сети Интернет», 2016, стр. 223-239.

[66]. Лаврищева Е.М. Теоретические основы моделирования программных систем из объектов и компонентов. Труды Международной научно-практической конференции «Теория активных систем», 2016, стр. 314-325.

[67]. Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП РАН, 2016, том 28, вып. 6, стр. 48-59. DOI: 10.15514/ISPRAS-2016-28(3)-12

[68]. Lavrischeva E. Assembling Paradigms of Programming in Software Engeneering. Journal of Software Engineering and Applications, vol. 9, no. 6, 2016, pp. 296-317.

[69]. Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle. In Proc. of the Science and Information Conference, 2015, pp. 965-972.

[70]. Е.М. Лаврищева, В.С. Мутилин, А.Г. Рыжов. Аспекты моделирования вариабельных программных и операционных систем. Труды. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», 2017, стр. 327-341.

[71]. Lavrischeva E.M., Mutilin V.S., Ryzhov A.G. Designing variability models for software, operating systems and their families. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 93-110, DOI: 10.15514/ISPRAS- 2017(5).

[72]. Lavrischeva E. Scientific Basis of System Programming. Journal of Software Engineering and Applications, No. 11, pp. 408-434.

[73]. E.M. Lavrischeva. The Scientific basis of Software Engineering. International Journal of Applied and Natural Sciences, vol. 7, issue 5, 2018, pp. 15-32.

[74]. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том.30, вып. 3, 2018, стр. 99-120. DOI: 10.15514/ISPRAS-2018-30(3)-8.

Метод анализа атак повторного использования кода★

[1] *А.В. Вишняков <vishnya@ispras.ru>*
[1] *А.Р. Нурмухаметов <oleshka@ispras.ru>*
[1] *Ш.Ф. Курмангалеев <kursh@ispras.ru>*
[1,2,3,4] *С.С. Гайсарян <ssg@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*
[2] *2119991 ГСП-1 Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК*
[3] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[4] *Национальный исследовательский университет «Высшая школа экономики» 101000, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. Обеспечение безопасности программного обеспечения является на сегодняшний день одной из первостепенных задач. Сбои в работе программного обеспечения могут привести к серьезным последствиям, а злонамеренная эксплуатация уязвимостей может причинить колоссальный ущерб. Крупные корпорации уделяют особое внимание анализу инцидентов информационной безопасности. Атаки повторного использования кода, основанные на возвратно-ориентированном программировании (ROP), приобретают всю большую популярность с каждым годом и могут быть применены даже в условиях работы защитных механизмов современных операционных систем. В отличие от обычного шелл-кода, где инструкции размещаются последовательно в памяти, ROP-цепочка состоит из множества маленьких блоков инструкций (гаджетов) и использует стек для связывания этих блоков, что затрудняет анализ ROP-эксплойтов. Целью данной работы является упрощение обратной инженерии ROP-эксплойтов. В этой статье предлагается метод анализа атак повторного использования кода, который позволяет восстановить семантику ROP-цепочки: разбить цепочку на гаджеты, определить семантику отдельных гаджетов и восстановить прототипы вызванных в ходе выполнения цепочки функций и системных вызовов и значения их аргументов. Семантика гаджета определяется его принадлежностью параметризованным типам. Каждый тип задается постусловием (булевым предикатом), которое должно быть всегда истинно после выполнения гаджета. Метод был реализован в виде программного инструмента и апробирован на реальных ROP-эксплойтах, найденных в интернете.

★ Работа поддержана грантом РФФИ № 17-01-00600

Ключевые слова: компьютерная безопасность; бинарный анализ; уязвимость; возвратно-ориентированное программирование; ROP; классификация гаджетов; атака повторного использования кода; инцидент информационной безопасности

DOI: 10.15514/ISPRAS-2018-30(5)-2

Для цитирования: Вишняков А.В, Нурмухаметов А.Р., Курмангалеев Ш.Ф., Гайсарян С.С. Метод анализа атак повторного использования кода. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 31-54. DOI: 10.15514/ISPRAS-2018-30(5)-2

1. Введение

Обеспечение безопасности программного обеспечения является на сегодняшний день одной из первостепенных задач. Программные продукты применяются в повседневно окружающих нас вещах: компьютерах, смартфонах, автомобилях, банкоматах, объектах городской инфраструктуры, медицинском оборудовании жизнеобеспечения и технологиях «интернета вещей». Сбои в работе программного обеспечения могут привести к серьезным последствиям: денежным убыткам, деградации средств коммуникации, задержке в работе экстренных служб, авариям и даже причинению вреда здоровью человека. А злонамеренная эксплуатация уязвимостей может причинить колоссальный ущерб. По данным Национального института стандартов и технологий США ежегодно публикуются тысячи описаний новых уязвимостей CVE (рис. 1) [1,2]. Крупные корпорации уделяют особое внимание анализу инцидентов информационной безопасности.

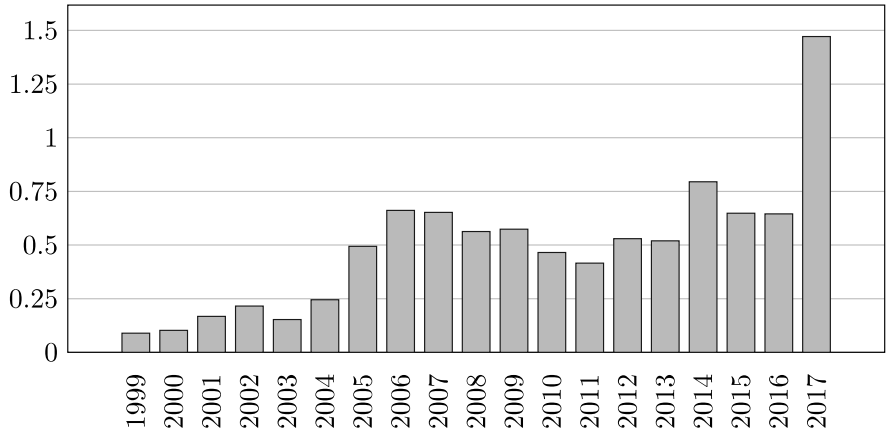

Рис. 1. Количество (десятки тысяч) новых уязвимостей (CVE) в год
Fig. 1. Tens of thousands of vulnerabilities (CVE) by year

Для эксплуатации уязвимостей в условиях работы защитных механизмов современных операционных систем часто применяется техника возвратно-ориентированного программирования (ROP). Это атака повторного

использования кода, позволяющая обходить защитный механизм, запрещающий региону памяти быть одновременно доступным на запись и исполнение (DEP), и современные реализации рандомизации размещения адресного пространства (ASLR), которые оставляют часть адресного пространства программы нерандомизированной. В частности, в Linux адрес загрузки кода программы часто остается постоянным, а некоторые динамические библиотеки Windows загружаются по фиксированным адресам. Злоумышленник использует кусочки кода из нерандомизированного адресного пространства программы, которые называются гаджетами. Каждый гаджет выполняет некоторые вычисления (например, складывает значения двух регистров) и передает управление следующему гаджету. Гаджеты связываются в цепочку последовательно выполняемых кусочков кода. Таким образом, с помощью цепочки гаджетов можно выполнить некоторые вредоносные действия.

Гаджет – это последовательность инструкций, которая заканчивается инструкцией передачи управления (`ret`). В отличие от обычной программы, инструкции ROP-цепочки не размещаются последовательно в памяти, а вместо этого разбиваются на маленькие гаджеты, связанные инструкциями, которые получают адрес следующего гаджета со стека. Такое стековое связывание инструкций затрудняет анализ ROP-цепочек.

Эксплойт – это программа, входные данные или последовательность команд, использующие уязвимость, чтобы добиться непредусмотренного поведения системы. Целью данной работы является упрощение обратной инженерии ROP-эксплойтов.

В этой статье предлагается метод анализа атак повторного использования кода, который позволяет восстановить семантику ROP-цепочки: разбить цепочку на гаджеты, определить семантику отдельных гаджетов и восстановить вызванные в ходе выполнения цепочки функции и системные вызовы и значения их аргументов.

Статья организована следующим образом. Во втором разделе приводится обзор атак и защитных механизмов, послуживших предпосылками к появлению ROP (подраздел 2.5). В третьем разделе проводится обзор существующих методов анализа ROP-атак. В четвертом разделе описывается предложенный метод анализа атак повторного использования кода. В пятом разделе рассматриваются детали реализации предлагаемого метода. Результаты практического применения приводятся в шестом разделе.

2. Обзор атак и защитных механизмов

В этом разделе приводится обзор атак на переполнение буфера на стеке. Описываются защитные механизмы операционной системы: ограничение исполняемых областей (DEP) и рандомизация размещения адресного пространства (ASLR). В разделе 2.5 дается определение возвратно-ориентированного программирования – метода эксплуатации уязвимости

переполнения буфера на стеке, позволяющего обойти DEP и современные реализации ASLR.

2.1 Уязвимость переполнения буфера на стеке

Уязвимость переполнения буфера на стеке возникает, когда размер данных, записываемых в буфер на стеке, превышает размер этого буфера [3]. Например, в приведенной ниже программе на Си уязвимая функция `vul` не проверяет длину сроки `str`, записываемой в буфер на стеке фиксированного размера `buf`. Если длина первого аргумента командной строки `argv[1]` окажется большей или равной размеру буфера `buf`, то произойдет переполнение буфера на стеке.

```
void vul(char *str) {
    char buf[512];
    strcpy(buf, str);
}
int main(int argc, char *argv[]) {
    vul(argv[1]);
    return 0;
}
```

На рисунке 2a показан стековый фрейм функции `vul` до переполнения. Стек в архитектуре x86 растет от больших адресов к меньшим (на рисунке – сверху вниз). Аргументы функции поочередно кладутся на стек справа налево. При вызове функции адрес возврата кладется на стек, после чего функция может сохранить старое значение регистра `ebp` и выделить на стеке память для локальных переменных, в нашем случае – для буфера `buf`. Данные в буфер записываются в порядке возрастания адресов (на рисунке – снизу вверх). Переполнение буфера приводит к перезаписи ячеек выше по стеку, в том числе адреса возврата, после чего почти всегда следует аварийное завершение программы.

Эксплуатация уязвимости переполнения буфера на стеке позволяет выполнять произвольный код. Рассмотрим ситуацию, когда злоумышленник контролирует значение первого аргумента командной строки `argv[1]`, а следовательно, контролирует значения, записываемые в буфер `buf`. В таком случае злоумышленник может добиться перезаписи адреса возврата указателем на размещенный на стеке вредоносный код (рис. 2b). Таким образом, после возврата из функции `vul` управление передастся на сформированный злоумышленником код. Обычно в качестве такого кода используется код, приводящий к вызову командной оболочки операционной системы, который называется шелл-кодом. Чтобы избежать негативных последствий от переполнения буфера на стеке, появились различные защитные механизмы.

2.2 Ограничение исполняемых областей (DEP)

Ограничение исполняемых областей (DEP) – защитный механизм операционной системы, запрещающий исполнение кода из областей памяти, помеченных как «данные». Попытка исполнения кода из помеченных областей вызывает исключение и влечет за собой аварийное завершение программы. В частности, стек и куча становятся недоступными для выполнения, что предотвращает выполнение размещенного на них вредоносного кода. Механизм успешно применяется в операционных системах Windows, Linux и др.

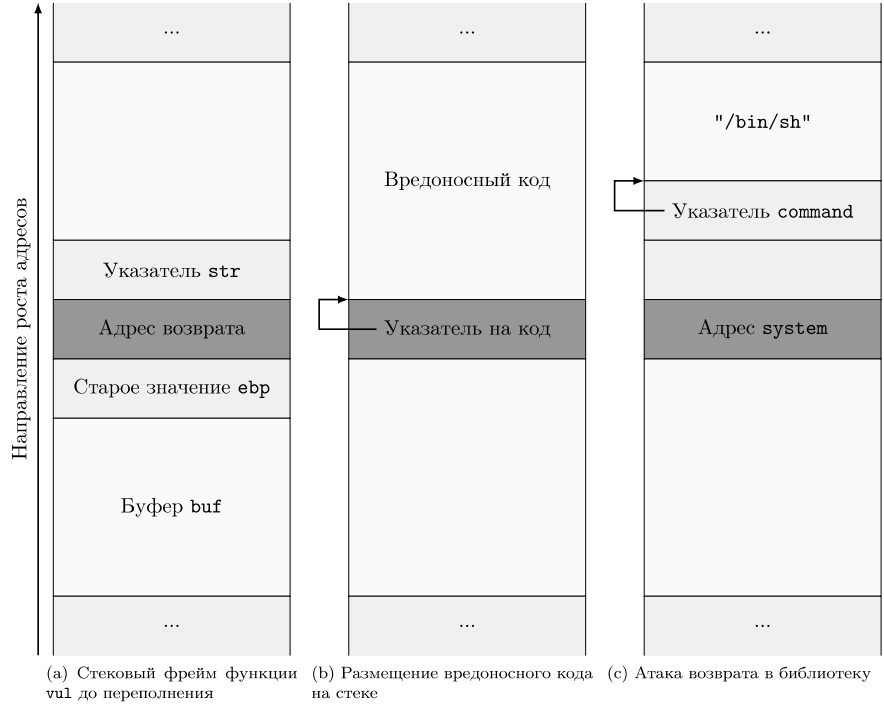

Рис. 2. Стековый фрейм функции `vul` *и способы эксплуатации уязвимости переполнения буфера* `buf` *на стеке*
Fig. 2. A stack frame and different buffer overflow exploitation techniques

2.3 Атака возврата в библиотеку

Для обхода DEP используется атака возврата в библиотеку. Атака заключается в подмене адреса возврата адресом некоторой библиотечной функции, например, функции system из библиотеки `libc`.

На рис. 2c показано состояние стека после переполнения. Адрес возврата перезаписан адресом функции `system(const char *command)`. Выше

лежит произвольный адрес возврата из функции system и ее единственный аргумент `command`, который является указателем на нуль-терминированную строку `"/bin/sh"`, размещенную следом за указателем. Таким образом, после возврата из функции `vul` вызовется библиотечная функция `system("/bin/sh")`, которая в свою очередь вызовет командную оболочку операционной системы.

2.4 Рандомизация размещения адресного пространства (ASLR)

Рандомизации размещения адресного пространства (ASLR) – защитный механизм операционной системы, позволяющий размещать ключевые элементы процесса (образ программы, стек, куча, динамические библиотеки) по различным адресам во время загрузки исполняемого файла. Данная защита затрудняет проведение атаки возврата в библиотеку, т.к. адрес библиотечной функции неизвестен до загрузки программы и отличается для каждого запуска.

Следует отметить, что рандомизация адресов исполняемых секций программы или библиотеки требует, чтобы они были скомпилированы в позиционно-независимый код, что не всегда выполняется. Так в Linux адрес загрузки кода программы часто остается постоянным, а некоторые динамические библиотеки Windows загружаются по фиксированным адресам. Таким образом, в условиях работы современных реализаций ASLR часть адресного пространства программы остается нерандомизированной.

2.5 Возвратно-ориентированное программирование (ROP)

Возвратно-ориентированное программирование (ROP) [4] – метод эксплуатации уязвимости переполнения буфера на стеке, который по сути является обобщением атаки возврата в библиотеку. Метод так же применим в условиях работы DEP, но представляет большую опасность, т.к. может быть использован для обхода современных реализаций ASLR, когда часть адресного пространства остается нерандомизированной (разд. 2.4).

ROP предполагает использование последовательностей инструкций в нерандомизированных исполняемых областях памяти, которые заканчиваются инструкцией передачи управления (`ret`). Такие последовательности инструкций называются гаджетами. Следует отметить, что архитектура x86 не требует выравнивания адресов инструкций, т.е. позволяет выполнение инструкций, размещенных по произвольным адресам памяти. А значит, некоторая последовательность инструкций в программе может содержать в себе гаджет, отсутствовавший в коде программы. Ниже приводятся бинарный и ассемблерный коды гаджета, который содержится внутри последовательности инструкций оригинальной программы.

```
f7c7070000000f9545c3 → test edi, 0x7 ;
```

```
                                setnz BYTE PTR [ebp-0x3d]
c7070000000f9545c3 → mov DWORD PTR [edi], 0xf000000 ;
                                xchg ebp, eax ; inc ebp ; ret
```

Гаджеты собираются в цепочки, а их адреса размещаются от адреса возврата на стеке так, чтобы первый гаджет передавал управление второму, второй – третьему и т.д. Таким образом, с помощью цепочки гаджетов можно выполнить некоторые вредоносные действия.

На рисунке 3 приводится состояние стека после размещения на нем ROP-цепочки, которая производит запись значения `memValue` по адресу `memAddr`. Адрес возврата перезаписан адресом первого гаджета. После возврата из функции, в которой произошло переполнение, управление передастся первому гаджету, который загрузит со стека значение `memValue` на регистр `eax`. При возврате (после выполнения инструкции `ret`) первый гаджет передаст управление второму гаджету, который в свою очередь загрузит значение `memAddr` на регистр `edx`. Потом третий гаджет сохранит значение регистра `eax` (`memValue`) по адресу `edx` (`memAddr`). Далее управление передастся четвертому гаджету и т.д.

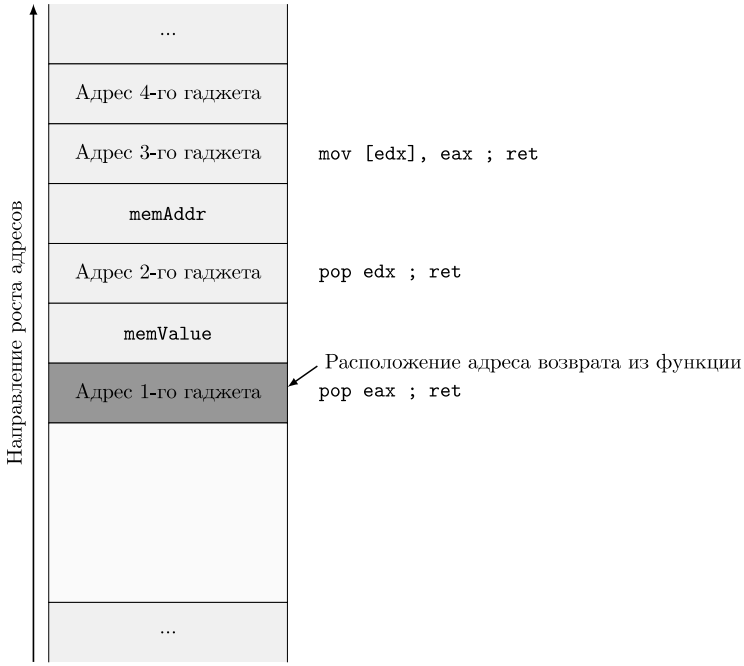

Рис. 3. Состояние стека после размещения на нем ROP-цепочки, которая производит запись значения `memValue` *по адресу* `memAddr`

Fig.3. A ROP chain, storing `memValue` *to* `memAddr`*, stack frame*

Ниже приводится эта же ROP-цепочка в бинарном виде, которая записывает значение `"/bin"` по адресу `0x0830caa0`. Эта последовательность байтов размещается злоумышленником на стеке от адреса возврата.

```
00000000  47 65 06 08 2f 62 69 6e  3d 76 07 08 a0 ca 30 08  |Ge../bin=v....0.|
00000010  b5 8b 08 08                                       |....|
00000014
```

3. Обзор существующих решений

В этом разделе описываются существующие решения проблем определения семантики гаджета и анализа атак повторного использования кода.

3.1 Определение семантики гаджета

Schwartz и др. [5] предложили определять функциональность гаджета его принадлежностью типам, которые приведены в таблице 1. Набор типов гаджетов задает новую архитектуру набора команд (ISA), в которой каждый тип гаджета исполняет роль инструкции. Семантика каждого типа гаджета определяется постусловием (булевым предикатом) $\mathscr{B}$, которое должно быть всегда истинно после выполнения гаджета.

Табл. 1. Типы гаджетов. `[Addr]` *означает доступ к памяти по адресу* `Addr`, `∘` *– бинарную операцию.* `a ← b` *означает, что конечное значение* `a` *равно начальному значению* `b`. `X ∘← Y` *– сокращение для* `X ← X ∘ Y`
Table 1. Gadget types

Тип	**Параметры**	**Определение семантики**
`NoOpG`	—	Не меняет ничего в памяти и на регистрах
`JumpG`	`AddrReg`	`IP ← AddrReg`
`MoveRegG`	`InReg`, `OutReg`	`OutReg ← InReg`
`LoadConstG`	`OutReg`, `Offset`	`OutReg ← [SP + Offset]`
`ArithmeticG`	`InReg1`, `InReg2`, `OutReg`, `∘`	`OutReg ← InReg1 ∘ InReg2`
`LoadMemG`	`AddrReg`, `OutReg`, `Offset`	`OutReg ← [AddrReg + Offset]`
`StoreMemG`	`AddrReg`, `InReg`, `Offset`	`[AddrReg + Offset] ← InReg`
`ArithmeticLoadG`	`AddrReg`, `OutReg`, `Offset`, `∘`	`OutReg ∘← [AddrReg + Offset]`
`ArithmeticStoreG`	`AddrReg`, `InReg`, `Offset`, `∘`	`[AddrReg + Offset] ∘← InReg`

Будем говорить, что последовательность инструкций $\mathscr{J}$ удовлетворяет постусловию $\mathscr{B}$, если для любого начального состояния после выполнения $\mathscr{J}$ постусловие $\mathscr{B}$ истинно. Начальное состояние состоит из присваиваний регистрам и памяти некоторых начальных значений.

Следует отметить, что один гаджет может принадлежать сразу нескольким типам. Например, гаджет `push eax ; pop ebx ; pop ecx ; ret` одновременно перемещает `eax` в `ebx` и загружает значение со стека в `ecx`, что

соответствует типам `MoveRegG: ebx ← eax` и `LoadConstG: ecx ← [esp + 0]`.

3.1.1 Семантический анализ

Для того чтобы определить, удовлетворяет ли последовательность инструкций $\mathscr{I}$ постусловию $\mathscr{B}$, Schwartz и др. [5] используют известную технику из формальной верификации – вычисление слабейшего предусловия [6]. Проще говоря, слабейшее предусловие $wp(\mathscr{I}, \mathscr{B})$ для последовательности инструкций $\mathscr{I}$ и постусловия $\mathscr{B}$ – это булево предусловие, которое описывает, когда $\mathscr{I}$ завершается в состоянии, удовлетворяющем $\mathscr{B}$. Слабейшие предусловия используются, чтобы убедиться, что определение семантики гаджета всегда выполняется после выполнения последовательности инструкций $\mathscr{I}$. Для этого достаточно проверить:

$$wp(\mathscr{I}, \mathscr{B}) \equiv true.$$

Если формула верна, то $\mathscr{B}$ всегда истинно после выполнения $\mathscr{I}$, а значит, $\mathscr{I}$ – гаджет с семантическим типом $\mathscr{B}$.

Однако формальная верификация гаджетов показала себя очень медленной на практике. Для ускорения процесса опредеделения, принадлежит ли гаджет тому или иному типу, инструкции гаджета предварительно несколько раз выполняются с использованием случайных входных данных, и проверяется истинность $\mathscr{B}$. Если $\mathscr{B}$ окажется ложным хотя бы для одного выполнения, то последовательность инструкций не может быть гаджетом этого типа. Таким образом, более сложное вычисление слабейшего предусловия производится, только если $\mathscr{B}$ истинно для всех выполнений.

Выполнение со случайными входными данными может быть также использовано для выявления возможных значений параметров (табл. 1) гаджетов. Например, посмотрев на значения регистров и на адреса чтения из памяти, можно вычислить набор возможных смещений (`Offset`) для гаджета загрузки из памяти `LoadMemG`.

3.2 deRop

В отличие от использования традиционного шелл-кода, который внедряется в память процесса, возвратно-ориентированное программирование позволяет производить произвольные вычисления, используя уже имеющийся в памяти код. Поэтому для анализа ROP-атак затруднительно использовать существующие традиционные инструменты анализа бинарного кода. Для решения озвученной проблемы был предложен инструмент deRop [7] – ROP-эксплойт приводится к семантически эквивалентному обычному шелл-коду, который уже может быть проанализирован существующими инструментами. Авторы преимущественно используют статический анализ и выделяют следующие трудности анализа ROP-атак:

Обнаружение гаджетов. При эксплуатации переполнения буфера на стеке перед адресом первого гаджета (которым будет перезаписан адрес возврата) записывается буфер произвольных незначительных данных. Более того, между адресом первого и второго гаджета также могут быть пропущены ячейки (например, когда функция, в которой происходит переполнение, чистит за собой аргументы со стека инструкций `ret n`). Несмотря на то что deRop пытается использовать статический анализ, насколько это возможно, избегая использование динамического анализа, обнаружение первых двух гаджетов производится с использованием отладчика.

Отслеживание указателя стека. В ROP-эксплойте указатель стека используется для получения адреса следующего гаджета так же, как указатель инструкции (счетчик команд) – для получения адреса следующей инструкции. Поэтому необходимо отслеживать указатель стека для обнаружения следующего гаджета.

Расположение стека и констант. Для загрузки констант в регистр в шелл-коде обычно используются `mov reg, imm`, в то время как в ROP обычно используются `pop reg`. Расположение стека в исходной ROP-цепочке отлично от расположения в выходном семантически эквивалентном ей шелл-коде. Поэтому необходимо отслеживать расположение констант на стеке.

Вызовы функций. Некоторые гаджеты в ROP-цепочке используются для вызова функций. Необходимо выявлять такие вызовы функций и вызывать их традиционным образом. Более того, необходимо определять значения аргументов функции (в т.ч. аргументов, которые являются константой или указателем) для каждого вызова.

Циклы. ROP-цепочка может содержать циклы. Необходимо уметь их обнаруживать и определять условие выхода из цикла.

3.2.1 Постобработка

Как только все гаджеты были проанализированы, производятся несколько этапов постобработки для упрощения выходного кода:

Данные в памяти. Вычисляются значения операндов инструкций, обращающихся к памяти, и операнды заменяются константами.

Нулевые байты. Обычно требуется, чтобы шелл-код не содержал нулевых байтов, т.к. это приводит к обрезанию шелл-кода после некоторых операций (например, `strcpy`). Данная проблема решается заменой всех нулевых байтов на ненулевые значения и добавлением декодера в начало шелл-кода, который восстановит оригинальные значения.

Адрес возврата. Адрес возврата в эксплойте заменяется адресом начала результирующего шелл-кода.

3.3 ROPMEMU

ROPMEMU [8] – фреймворк для анализа сложных атак повторного использования кода. Авторы используют динамический подход к анализу бинарного кода и выделяют следующие проблемы анализа ROP-атак (C1—C3 уже были упомянуты в подразделе 3.2):

[C1] Избыточность – большинство ROP-гаджетов содержат лишние инструкции. Например, гаджет, предназначенный для инкрементирования `eax`, может также загружать (`pop`) значение со стека до передачи управления следующему гаджету (`ret`).

[C2] Стековое связывание инструкций – в отличие от обычной программы, где инструкции размещаются последовательно в памяти, ROP-эксплойт разбивается на маленькие гаджеты, связанные в цепочку инструкциями косвенной передачи управления (`ret`).

[C3] Нехватка значений констант – ROP-цепочки обычно составляются из параметризованных гаджетов (например, загрузки произвольного значения в регистр `rax`), которые используют параметры, сохраненные на стеке.

[C4] Условные ветвления – в отличие от традиционного изменения указателя инструкции (счетчика команд) условное ветвление в ROP-цепочке изменяет указатель стека. Таким образом, простой условный переход реализизуется несколькими гаджетами (стр. 18—19 [9]). Для приведения цепочки к более читаемому коду необходимо распознавать такие условные ветвления и заменять их одной инструкцией ветвления.

[C5] Возврат в функции – вызовы функций в ROP обычно реализуются простым возвратом (`ret`) во входную точку функции. Т.к. обычные гаджеты также часто берутся из кода, расположенного внутри библиотек, необходимо уметь отличать вызов функции от очередного гаджета.

[C6] Динамически генерируемые цепочки – ROP-цепочка не обязательно сразу целиком размещается в памяти, а могут быть использованы гаджеты, которые подготовят выполнение других гаджетов в будущем.

[C7] Условие останова – авторы предполагают, что аналитик способен определить начало ROP-цепочки в памяти. Однако необходимо завершить процесс эмуляции, когда все гаджеты были извлечены.

В фреймворке ROPMEMU используется набор различных техник для анализа ROP-цепочек и восстановления эквивалентного им кода в форме, которая может быть проанализирована традиционными инструментами обратной инженерии, такими как IDA Pro [10]. Предполагается, что у аналитика имеются дамп памяти и входная точка (первой) ROP-цепочки в нем. Оставшиеся динамически генерируемые цепочки восстанавливаются самим фреймворком, который имеет пять основных фаз анализа.

3.3.1 Многопутевая эмуляция

На этом шаге эмулируются ассемблерные инструкции, из которых состоит ROP-цепочка (C2). Исследуются все возможные ветвления и для каждого пути выполнения генерируется независимая трасса (аннотированная значениями регистров и памяти). Эмулятор также распознает возвраты в библиотечные функции, пропускает их тело и симулирует их выполнение, генерируя фиктивные данные и возвращаемое значение (C5).

Эмулятор изначально считывает содержимое памяти из дампа памяти и поддерживает теневую память [11]. Условие останова (C7) определяется набором эвристик, основанных на принципе локальности (эмулятор обнаруживает большое относительное изменение указателя стека) и длине гаджета, исключая обнаруженные вызовы функций. Как только срабатывает условие останова, содержимое теневой памяти и трасса выполнения сохраняются на диск и исследуются на предмет наличия новых ROP-цепочек. Если таковые найдены, эмулятор перезапускается, чтобы проанализировать следующую цепочку, и так до тех пор, пока все динамически генерируемые цепочки не будут обнаружены и проанализированы (C6).

Для ROP-цепочек со сложным потоком управления, простой подход, основанный на эмуляции, не достаточен для анализа всего ROP-эксплойта. Ведь покрытие ограничено только выполненными условными переходами, которые часто зависят от фиктивных возвращаемых значений функций, сгенерированных эмулятором. Данную проблему решает многопутевая эмуляция, которая является адаптированной к ROP-цепочкам версией алгоритма многопутевого выполнения [12]. В частности, эмулятор распознает, когда указатель стека изменяется в зависимости от содержимого регистра флагов. В конце процесса эмуляции из трассы получается список всех точек ветвления вместе со значениями флагов в каждой из них. Далее эмулятор перезапускается с указанием инвертировать переход в точке ветвления. Таким образом, выполнение пройдет по другому пути. Исследование ветвлений прекращается, когда все ветви проанализированы.

Однако, при наличии циклов в ROP-цепочке, эмулятор может застрять в бесконечном пути выполнения. Для решения этой проблемы отслеживается число повторений указателя стека во время выполнения инструкций ветвления. Если это число превосходит некоторый допустимый порог, эмулятор инвертирует переход, чтобы насильно прекратить цикл и исследовать оставшуюся часть графа потока управления.

3.3.2 Разбиение трассы

На этой фазе анализируются все сгенерированные эмулятором трассы, удаляются повторения и извлекаются уникальные блоки кода. Каждая трасса разрезается в каждой точке ветвления, генерируется новый блок, который

сохраняется в отдельную трассу. В результате, будет получен набор трасс, ассоциированных с каждым «базовым блоком» в цепочке.

3.3.3 Отвязывание инструкций от стека

На этом этапе из трассы удаляются все инструкции безусловной передачи управления (`ret`, `call`, `jmp`) и содержимое последовательно выполняемых гаджетов сливается в один базовый блок (C2). Затем инструкции mov упрощаются, благодаря вычислению их операндов (например, `mov rax, [rsp + 0x30]`). Инструкции `pop` заменяются инструкциями `mov`, все необходимые значения получаются из соответствующих ячеек на стеке (C3).

3.3.4 Восстановление графа потока управления

На этом проходе все трассы сливаются в единое графовое представление. Потом граф транслируется в настоящую программу x86, благодаря распознаванию инструкций, ассоциированных с условными ветвлениями, и замене их традиционными, использующими указатель инструкции (счетчик команд) условными переходами (C4).

Следующей задачей данного прохода является обнаружение и сворачивание циклов. ROP-цепочки могут содержать как возвратно-ориентированные циклы, так и развернутые циклы. В первом случае ROP-инструкции используются для повторения одного и того же блока гаджетов на стеке с выходом по условию. Развернутые циклы в свою очередь повторяют одну и ту же последовательность гаджетов заранее определенное (константное) количество раз. Фреймворк автоматически определяет рекуррентные паттерны и заменяет их более компактным куском ассемблерного кода, представляющим из себя цикл с той же семантикой.

Полученный в результате код оборачивается рабочими прологом и эпилогом функции и включается в отдельный ELF файл, чтобы позволить использовать традиционные инструменты обратной инженериии (например, IDA Pro [10]) для работы с ним.

3.3.5 Бинарная оптимизация

На заключительном шаге применяются известные компиляторные преобразования для дальнейшего упрощения ассемблерного кода в ELF файле. В частности, удаляется мертвый код, применяются преобразования, описанные в п. 3.2.1, и генерируется чистая и оптимизированная версия эксплойта (C1).

4. Метод анализа атак повторного использования кода

Предлагаемый в данной статье метод анализа атак повторного использования кода позволяет восстановить семантику ROP цепочки и проследить за ходом атаки. По бинарной ROP-цепочке восстанавливается последовательность

вызванных гаджетов. Найденные гаджеты классифицируются по семантическим типам, и определяются значения параметров гаджетов. Помимо того в цепочке выявляются вызовы функций и системные вызовы, восстанавливаются их прототипы и значения аргументов. Следует отметить, что в данной работе не ставится задача проанализировать все пути выполнения ROP-цепочки, а достаточно разбора хотя бы одного из них. Поэтому предлагаемый метод не учитывает условные ветвления в ROP-цепочках.

4.1 Фрейм гаджета

Для декомпозиции бинарной ROP-цепочки на гаджеты вводится понятие фрейма гаджета аналогичное стековому кадру x86. Цепочка гаджетов разбивается на фреймы. Фрейм гаджета содержит в себе значения параметров гаджета (например, значение, загружаемое на регистр со стека гаджетом `LoadConstG`) и адрес следующего гаджета. Начало фрейма определяется значением указателя стека перед выполнением первой инструкции гаджета.

Рис. 4. Фрейм гаджета `pop eax ; ret 8`
Fig.4. `pop eax ; ret 8` *gadget frame*

На рисунке 4 фигурной скобкой обозначены гараницы фрейма гаджета `pop eax ; ret 8`. Гаджет загружает значение со стека в `eax`, что соответствует типу загрузки константы `LoadConstG: eax ← [esp + 0]`. Гаджет имеет размер фрейма `FrameSize = 16`, а адрес следующего гаджета располагается по смещению 4 от начала фрейма (`NextAddr = [esp + 4]`).

4.2 Классификация гаджетов

Классификации гаджетов [13] позволяет определить семантику гаджетов. Семантика гаджета определяется набором булевых постусловий (типов гаджета) и значениями их параметров, которым удовлетворяют инструкции

гаджета (разд. 3.1). Для анализа ROP-цепочек, найденных в интернете, предложенного Schwartz и др. [5] набора типов гаджетов (табл. 1) оказалось недостаточно, и он был расширен дополнительными типами, которые приводятся в таблице 2. Более того, были добавлены типы гаджетов, которые не гарантируют сохранения управления (внизу таблицы 2).

Табл. 2. Дополнительные типы гаджетов. `[Addr]` *означает доступ к памяти по адресу* `Addr`, `∘` *– бинарную операцию.* `a ← b` *означает, что конечное значение* `a` *равно начальному значению* `b`. `X ∘← Y` *– сокращение для* `X ← X ∘ Y`
Table 2. Extended gadget types

Тип	**Параметры**	**Определение семантики**
`JumpMemG`	`AddrReg`, `Offset`	`IP ← [AddrReg + Offset]`
`GetSPG`	`OutReg`	`OutReg ← SP`
`InitConstG`	`OutReg`, `Value`	`OutReg ← Value`
`InitMemG`	`AddrReg`, `Value`, `Offset`, `Size`	`[AddrReg + Offset] ← Value`
`NegG`	`InReg`, `OutReg`	`OutReg ← −InReg`
`ArithmeticConstG`	`InReg`, `OutReg`, `Value`, `∘` (+/⊕)	`OutReg ← InReg ∘ Value`
`ShiftStackG`	`Offset`, `∘` (+/−)	`SP ∘← Offset`

`PushAllG`	—	`([ESP − 4] ← EAX) ∧`
		`([ESP − 8] ← ECX) ∧`
		`([ESP − 12] ← EDX) ∧`
		`([ESP − 16] ← EBX) ∧`
		`([ESP − 20] ← ESP) ∧`
		`([ESP − 24] ← EBP) ∧`
		`([ESP − 28] ← ESI) ∧`
		`(EIP ← EDI)`

Не сохраняют управление		
`JumpSPG`	—	`IP ← SP`
`CallG`	`AddrReg`	`IP ← AddrReg`
`CallMemG`	`AddrReg`, `Offset`	`IP ← [AddrReg + Offset]`
`IntG`	`Value`	Вызвать прерывание `Value`
`SyscallG`	—	Системный вызов

Классификация гаджета производится на основе анализа эффектов выполнения гаджета на случайных входных данных. Инструкции гаджета транслируются в промежуточное представление. Далее запускается процесс интерпретации промежуточного представления. Во время интерпретации отслеживаются обращения к регистрам и памяти. Если происходит первое чтение регистра или области памяти, считанное значение генерируется случайным образом. В результате интерпретации будут получены начальные и конечные значения регистров и памяти. На основе этой информации делается вывод о возможной принадлежности гаджета тому или иному типу. Например, для принадлежности типу `MoveRegG` должна существовать такая пара

регистров, что начальное значение первого регистра равно конечному значению второго. В результате анализа составляется список всех удовлетворяющих гаджету типов и их параметров (список кандидатов). Затем производится еще несколько запусков процесса интерпретации с отличными входными данными, в результате которых из списка кандидатов удаляются ошибочно определенные типы.

В результате классификации гаджета будут получены семантические типы гаджета и их параметры, а также информация о фрейме гаджета (разд. 4.1) – размер фрейма (`FrameSize`) и смещение ячейки с адресом следующего гаджета относительно начала фрейма (`NextAddr`).

Следует отметить, что классификация гаджета производится в результате выполнения гаджета на ограниченном количестве наборов конкретных входных данных, что в общем случае не гарантирует соответствия семантике результата выполнения гаджета на произвольных входных данных. Для точной классификации необходимо производить формальную верификацию семантики гаджета, как описывается в подразделе 3.1.1. Таким образом, возможна неверная классификация гаджета. Однако доля неверно классифицированных гаджетов после 10 запусков на случайных входных данных незначительна, что является приемлемым для задачи восстановления семантики ROP-цепочек.

4.3 Восстановление семантики ROP-цепочек

Бинарная ROP-цепочка загружается на теневой стек. Используя информацию о фрейме предыдущего гаджета, полученную в результате классификации, один за другим классифицируются гаджеты в цепочке. Смещение ячейки с адресом следующего гаджета относительно начала фрейма и размер фрейма по сути показывают, где брать адрес следующего гаджета для классификации и где начинается его фрейм соответственно. Указатель теневого стека всегда указывает на начало фрейма последнего классифицированного гаджета.

Для восстановления значений регистров и памяти перед выполнением гаджета (например, для восстановления аргументов системного вызова или функции) поддерживается общая для всех гаджетов теневая память [11]. Изначально теневая память пуста. Последовательно для каждого классифицированного гаджета цепочки производится несколько запусков процесса интерпретации его промежуточного представления с теневой памятью, выступающей в качестве начальных значений регистров и памяти. Считанные регистры и память, не содержащиеся в теневой памяти, генерируются случайном образом при каждом запуске интерпретации. Конечные значения регистров и памяти, которые не менялись от запуска к запуску, обновляются в теневой памяти.

Значения всех загружаемых ROP-цепочкой констант могут быть восстановлены из теневого стека. Одной лишь классификации гаджетов для этого недостаточно, т.к. она не учитывает данные, расположенные на теневом стеке, а генерирует считанные со стека значения случайным образом.

Классификация гаджета загрузки константы `LoadConstG` позволяет определить регистр `OutReg`, на который производится загрузка константы, и смещение `Offset`, по которому происходит чтение значения константы со стека. После классификации гаджета `LoadConstG` в теневую память добавляется значение регистра `OutReg`, загруженное с теневого стека по смещению `Offset` от указателя теневого стека.

Для обхода DEP в 32-битных Windows программах часто используется гаджет `PushAllG` (`pushad ; ret`), при помощи которого вызывается функция WinAPI `VirtualProtect` [14] (которая сделает стек исполняемым) и передается управление обычному шелл-коду, размещенному выше на стеке (рис. 5). Дело в том, что инструкция `pushad` сохраняет регистры общего назначения на стек. Если предварительно проинициализировать регистры соответствующими значениями, то на стеке окажется обыкновенная ROP-цепочка.

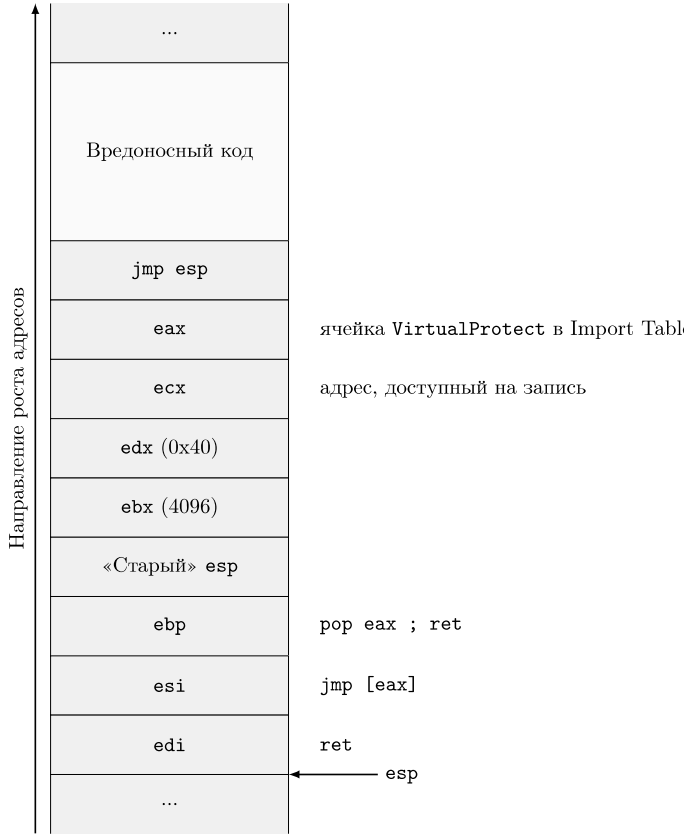

Рис. 5. Состояние стека после выполнения инструкции `pushad`
Fig.5. A `pushad` *stack frame*

Таким образом, сначала на регистр edi загружается адрес `NoOpG` гаджета (`ret`), на `esi` – адрес гаджета, который вызовет функцию `VirtualProtect` (например, `jmp [eax]`, при этом на `eax` предварительно загружается адрес ячейки `VirtualProtect` в таблице импортированных символов). На регистры `ebx`, `edx` и `ecx` загружаются 2—4 аргументы `VirtualProtect` соответственно. А в качестве адреса возврата из функции `VirtualProtect` на регистр `ebp` загружается адрес `ShiftStackG` гаджета (`pop eax ; ret`), инкрементирующего указатель стека. После выполнения инструкции `pushad` значения этих регистров будут лежать на стеке, как изображено на рисунке 5. В свою очередь выполнение инструкция возврата ret передаст управление по адресу гаджета, записанному в последний сохраненный регистр `edi` (`ret`). Далее управление передастся гаджету, который вызовет `VirtualProtect(esp, ebx, edx, ecx)`. А после возврата из функции `VirtualProtect` стек станет исполняемым и управление передастся гаджету, чей адрес был предварительно загружен на регистр `ebp` (`pop eax ; ret`). В результате, вызовется гаджет `JumpSPG` (`jmp esp`), который передаст управление обычному шелл-коду, размещенному сразу же выше по стеку и доступному теперь на исполнение.

Для такого гаджета `PushAllG` на теневой стек записываются соответсвующие значения регистров. Гаджет `JumpSPG` в свою очередь интерпретируется как передача управления обычному шелл-коду, размещенному на стеке, и производится дизассемблирование его байтов.

Следует отметить, что ROP-цепочка может предварительно записать гаджет в память, чтобы потом его использовать. Ниже приводится пример такой цепочки. Сначала в регистр `edx` загружается машинный код гаджета `mov [eax + ebp * 4], ebx ; ret`. Затем этот гаджет сохраняется в память по адресу `eax`. Далее загружаются параметры гаджета: `ebx` и `ebp`. Наконец передается управление по адресу `eax`, куда был предварительно сохранен гаджет, записывающий в память значение регистра `ebx` по адресу `eax + ebp * 4`.

```
pop edx ; ret  // edx = "\x89\x1c\xa8\xc3"
mov [eax], edx ; ret
pop ebx ; pop ebp ; ret
jmp eax  // mov [eax + ebp * 4], ebx ; ret
```

Таким образом, если во время разбора ROP-цепочки адрес следующего гаджета содержится в теневой памяти, то классифицируется гаджет из теневой памяти. Если же после классификации окажется, что этот гаджет нельзя отнести ни к одному из типов, то считается, что это передача управления обычному шелл-коду, предварительно сохраненному в память. Байты шелл-кода из теневой памяти также дизассемблируются.

4.3.1 Восстановление функций и системных вызовов

Функция может быть вызвана из ROP-цепочки с использованием гаджетов `JumpG`, `JumpMemG`, `CallG`, `CallMemG`, или же ее адрес может быть просто размещен на стеке. Системный вызов выполняется гажетом `IntG` в 32-разрядной операционной системе, а гаджетом `SyscallG` в 64-разрядной. Номер системного вызова, а также значения аргументов функции и системного вызова восстанавливаются из теневой памяти. Если по адресу аргумента в теневой памяти располагается нуль-терменированная строка, то она тоже восстанавливается.

Для ROP-цепочек под Linux по номеру системного вызова получается его имя. Имена вызванных функций можно восстановить, если функция была вызвана по адресу, считанному из таблицы импортированных символов (GOT в ELF и IAT в PE). Прототипы функций и системных вызовов Linux ищутся в man-pages [15], а прототипы функций Windows – в API Monitor [16].

5. Программная реализация

Описанный метод был реализован в виде программного инструмента. Инструмент получает на вход бинарную ROP-цепочку и исполняемый файл, в котором содержатся использованные в цепочке гаджеты. Следует отметить, что в данной работе не ставится задача поиска ROP-цепочки в эксплойте. Решение этой задачи возлагается на аналитика. Поддерживаются следующие форматы исполняемых файлов: ELF32, ELF64, PE32 и PE32+. Цепочки, использующие гаджеты из разных исполняемых файлов, в данный момент не поддерживаются.

5.1 Интерпретация промежуточного представления инструкций гаджета

В данной работе используется разработанное в ИСП РАН промежуточное представление инструкций [17], удовлетворяющее SSA-форме и имеющее трехадресный код. Адресные пространства памяти и регистров представляются в виде двух байтовых массивов. Адресное пространство регистров состоит из всех регистров машины с учетом наложений и пересечений. Для учета побочных эффектов используется слово состояния, аналогичное регистру флагов x86.

Инструкции гаджета транслируются в промежуточное представление, интерпретация которого позволяет получить начальные и конечные значения регистров и памяти. Изначально для каждого адресного пространства карты считанных и сохраненных значений пусты. Инструкции промежуточного представления заменяются эквивалентными блоками инструкций x86-64. При этом инструкции сохранения (STORE) заменяются на вызовы функции, обновляющей карту сохраненных значений. А инструкции чтения (LOAD)

заменяются вызовом функции, возвращающей актуальное значение, которая выполняет одно из следующих действий:

- считывает значение из карты сохраненных значений, если оно там присутствует;
- считывает значение из карты считанных значений, если оно там присутствует и отсутствует в карте сохраненных значений;
- добавляет в карту считанных значений случайно сгенерированное значение при первом обращении по адресу.

Далее производиться выполнение полученного x86-64 кода. В результате будут получены начальное и конечное состояния адресных пространств.

5.2 Разбор ROP-цепочки

Эмулируется загрузка исполняемого файла в виртуальное адресное пространство с разрешением релокаций. По адресу каждого гаджета в загруженном исполняемом файле дизассемблируются инструкции до инструкции передачи управления. Полученные инструкции транслируются в промежуточное представление и классифицируются. Аргументы функций и системных вызовов восстанавливаются согласно соглашению о вызове из теневой памяти, а в качестве возвращаемого значения функции в теневую память добавляется некоторое фиктивное значение. Далее происходит обновление теневого стека и теневой памяти, как описано в разделе 4.3.

В результате, будет получен текстовый файл, в котором будут перечислены последовательно вызванные гаджеты, а также их типы и параметры. Более того, будут приведены прототипы вызванных функций и системных вызовов с восстановленными значениями аргументов. Если ROP-эксплойт завершается вызовом обычного шелл-кода, то будут выведены его дизассемблированные инструкции.

6. Результаты практического применения

Предложенный в этой статье метод анализа атак повторного использования кода был апробирован на реальных ROP-эксплойтах, найденных в интернете. Бинарные ROP-цепочки извлекались вручную. Затем определялся исполняемый файл, из которого использовались гаджеты в цепочке.

Хорошими отправными точками для поиска ROP-эксплойтов послужили фреймворк для тестирования на проникновение Metasploit [18] и открытая база данных эксплойтов EDB [19]. К сожалению, исполняемый файл, из которого собирались гаджеты в цепочку, редко прилагается к эксплойту. В лучшем случае будут указаны версия программы, операционная система и/или дистрибутив. Поэтому исполняемые файлы часто приходится искать вручную и проверять, находятся ли по тем же адресам заявленные в эксплойте гаджеты. Для поиска старых версий пакетов дистрибутива Debian существует проект snapshot.debian.org [20], который несколько раз в день сохраняет текущее

состояние дистрибутива Debian, что значительно упрощает задачу поиска старых версий пакетов.

В табл. 3 приводится список ROP-эксплойтов, которые были успешно проанализированы реализованным инструментом анализа ROP-цепочек. Время анализа не превосходило пары секунд.

Табл. 3. Список проанализированных ROP-эксплойтов
Table 3. Analyzed ROP exploits

Приложение	Номер CVE	Платформа	Гаджеты из
MongoDB	CVE-2013-1892	Linux x86	mongod
Nagios3	CVE-2012-6096	Linux x86	history.cgi
ProFTPd	CVE-2010-4221	Linux x86	proftpd
Nginx	CVE-2013-2028	Linux x64	nginx
AbsoluteFTP	CVE-2011-5164	Windows x86	MFC42.dll
ComSndFTP	N/A 2012-06-08	Windows x86	msvcrt.dll

7. Заключение

В данной статье был предложен метод анализа атак повторного использования кода, который был реализован в виде программного инструмента. Разработанный метод позволяет упростить для аналитика задачу обратной инженерии ROP-эксплойтов. По бинарной ROP-цепочке восстанавливается список вызванных гаджетов и описывается их функциональность семантически с помощью булевого предиката, который должен быть всегда истинным после выполнения гаджета. Более того, восстанавливаются прототипы и значения аргументов вызванных функций и системных вызовов. Таким образом, аналитик может автоматизированно получить представление о семантике ROP-цепочки. Реализованный метод был апробирован на реальных ROP-эксплойтах, найденных в интернете.

Метод основывается на динамической интерпретации промежуточного представления инструкций ROP-цепочки. Семантика гаджета определяется в результате анализа эффектов выполнения гаджета на различных случайных входных данных. Для восстановления значений аргументов функций и системных вызовов во время анализа поддерживается теневая память.

Перспективным направлением для дальнейших работ является поддержка анализа условных переходов и циклов в ROP-цепочках. А для повышения точности определения семантики гаджета можно использовать известные техники формальной верификации. Также технической задачей является поддержка анализа ROP-цепочек, использующих гаджеты сразу из нескольких исполняемых файлов.

Список литературы

[1]. Common Vulnerabilities and Exposures (CVE). Available at: https://cve.mitre.org, accepted 10.11.2008.
[2]. Статистика уязвимостей (CVE) по годам. Available at: https://www.cvedetails.com/browse-by-date.php, accepted 10.11.2008.
[3]. CWE-121: Stack-based Buffer Overflow. Available at: https://cwe.mitre.org/data/definitions/121.html, accepted 10.11.2008.
[4]. Shacham H. The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86). In Proc. of the 14th ACM Conference on Computer and Communications Security, CCS'07, 2007, pp. 552–561.
[5]. Schwartz E.J, Avgerinos T., Brumley D. Q: Exploit Hardening Made Easy. In Proc. of the 20th USENIX Conference on Security, SEC'11, 2011, p. 25.
[6]. Jager I., Brumley D. Efficient Directionless Weakest Preconditions. Technical Report CMU-CyLab-10-002, 2010.
[7]. Lu K., Zou D., Wen W., Gao D. deRop: Removing Return-oriented Programming from Malware. In Proc. of the 27th Annual Computer Security Applications Conference, ACSAC'11, 2011, pp. 363–372.
[8]. Graziano M., Balzarotti D., Zidouemba A. ROPMEMU: A Framework for the Analysis of Complex Code-Reuse Attacks. In Proc. of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS'16, 2016, pp. 47–58.
[9]. Roemer R., Buchanan E., Shacham H., Savage S. Return-Oriented Programming: Systems, Languages, and Applications. ACM Transactions on Information and System Security, vol. 15, no. 1, 2012, pp. 2:1–2:34.
[10]. Инструмент IDA Pro. Available at: https://www.hex-rays.com/products/ida/, accepted 10.11.2008.
[11]. Nethercote N., Seward J. How to Shadow Every Byte of Memory Used by a Program. In Proc. of the 3rd International Conference on Virtual Execution Environments, VEE'07, 2007, pp. 65–74.
[12]. Moser A., Kruegel C., Kirda E. Exploring Multiple Execution Paths for Malware Analysis. In Proc. of the 2007 IEEE Symposium on Security and Privacy, SP'07, 2007, pp. 231–245.
[13]. Вишняков А.В. Классификация ROP гаджетов. Труды ИСП РАН, том 28, вып. 6, 2016 г., стр. 27–36. DOI: 10.15514/ISPRAS-2016-28(6)-2
[14]. VirtualProtect function (Windows). Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/aa366898(v=vs.85).aspx, accepted 10.11.2008.
[15]. The Linux man-pages project. Available at: https://www.kernel.org/doc/man-pages/.
[16]. API Monitor: Spy on API Calls and COM Interfaces. Available at: http://www.rohitab.com/apimonitor, accepted 10.11.2008.
[17]. Падарян В.А., Соловьев М.А., Кононов А.И. Моделирование операционной семантики машинных инструкций. Труды ИСП РАН, том 19, 2010 г., стр. 165–186.
[18]. Metasploit Framework. Available at: https://github.com/rapid7/metasploit-framework, accepted 10.11.2008.
[19]. Exploit Database. Available at: https://www.exploit-db.com, accepted 10.11.2008.
[20]. snapshot.debian.org. Available at: http://snapshot.debian.org, accepted 10.11.2008.

Method for analysis of code-reuse attacks

[1] *A.V. Vishnyakov <vishnya@ispras.ru>*
[1] *A.R. Nurmukhametov <oleshka@ispras.ru>*
[1] *Sh.F. Kurmagaleev <kursh@ispras.ru>*
[1,2,3,4] *S.S. Gaisaryan <ssg@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *Moscow Institute of Physics and Technology (State University) 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[4] *National Research University Higher School of Economics (HSE) 11 Myasnitskaya Ulitsa, Moscow, 101000, Russia*

Abstract. Providing security for computer programs is one of the paramount tasks nowadays. Failures in operation of program software can lead to serious consequences and exploitation of vulnerabilities can inflict immense harm. Large corporations pay particular attention to the analysis of computer security incidents. Code-reuse attacks based on return-oriented programming are gaining more and more popularity each year and can bypass even modern operating system protections. Unlike common shellcode, where instructions are placed consequently in memory, ROP chain contains of several small instruction blocks (gadgets) and uses stack to chain them together, which makes analysis of ROP exploits more difficult. The main goal of this work is to simplify reverse engineering of ROP exploits. In this paper I propose the method for analysis of code-reuse attacks, which allows one to split chain into gadgets, restore the semantics of each particular gadget, and restore prototypes and parameters values of system calls and functions called during the execution of ROP chain. Parametrized types define gadget semantics. Each gadget type is defined by a postcondition (boolean predicate) that must always be true after executing the gadget. The proposed method was implemented as a program tool and tested on real ROP exploits found on the internet.

Keywords: computer security; binary analysis; vulnerability; return-oriented programming; ROP; gadgets classification; code-reuse attack; computer security incident.

DOI: 10.15514/ISPRAS-2018-30(5)-2

For citation: Vishnyakov A.V., Nurmukhametov A.R., Kurmangaleev Sh.F., Gaisaryan S.S. Method for analysis of code-reuse attacks. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 31-54 (in Russian). DOI: 10.15514/ISPRAS-2018-30(5)-2

References

[1]. Common Vulnerabilities and Exposures (CVE). Режим доступа: https://cve.mitre.org, дата обращения 10.11.2008.
[2]. Статистика уязвимостей (CVE) по годам. Режим доступа: https://www.cvedetails.com/browse-by-date.php, дата обращения 10.11.2008.

[3]. CWE-121: Stack-based Buffer Overflow. Режим доступа: https://cwe.mitre.org/data/definitions/121.html, дата обращения 10.11.2008.
[4]. Shacham H. The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86). In Proc. of the 14th ACM Conference on Computer and Communications Security, CCS'07, 2007, pp. 552–561.
[5]. Schwartz E.J, Avgerinos T., Brumley D. Q: Exploit Hardening Made Easy. In Proc. of the 20th USENIX Conference on Security, SEC'11, 2011, p. 25.
[6]. Jager I., Brumley D. Efficient Directionless Weakest Preconditions. Technical Report CMU-CyLab-10-002, 2010.
[7]. Lu K., Zou D., Wen W., Gao D. deRop: Removing Return-oriented Programming from Malware. In Proc. of the 27th Annual Computer Security Applications Conference, ACSAC'11, 2011, pp. 363–372.
[8]. Graziano M., Balzarotti D., Zidouemba A. ROPMEMU: A Framework for the Analysis of Complex Code-Reuse Attacks. In Proc. of the 11th ACM on Asia Conference on Computer and Communications Security, ASIA CCS'16, 2016, pp. 47–58.
[9]. Roemer R., Buchanan E., Shacham H., Savage S. Return-Oriented Programming: Systems, Languages, and Applications. ACM Transactions on Information and System Security, vol. 15, no. 1, 2012, pp. 2:1–2:34.
[10]. Инструмент IDA Pro. Режим доступа: https://www.hex-rays.com/products/ida/, дата обращения 10.11.2008.
[11]. Nethercote N., Seward J. How to Shadow Every Byte of Memory Used by a Program. In Proc. of the 3rd International Conference on Virtual Execution Environments, VEE'07, 2007, pp. 65–74.
[12]. Moser A., Kruegel C., Kirda E. Exploring Multiple Execution Paths for Malware Analysis. In Proc. of the 2007 IEEE Symposium on Security and Privacy, SP'07, 2007, pp. 231–245.
[13]. Vishnyakov A.V. Classification of ROP gadgets. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 27–36 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-2
[14]. VirtualProtect function (Windows). https://msdn.microsoft.com/en-us/library/windows/desktop/aa366898(v=vs.85).aspx.
[15]. The Linux man-pages project. Режим доступа: https://www.kernel.org/doc/man-pages/.
[16]. API Monitor: Spy on API Calls and COM Interfaces. Режим доступа: http://www.rohitab.com/apimonitor, дата обращения 10.11.2008.
[17]. Padaryan V.A., Soloviev M.A., Kononov A.I. Modeling operational semantics of machine instructions. Trudy ISP RAN/Proc. ISP RAS, 2010, vol. 19, pp. 165–186 (in Russian).
[18]. Metasploit Framework. Режим доступа: https://github.com/rapid7/metasploit-framework, дата обращения 10.11.2008.
[19]. Exploit Database. Режим доступа: https://www.exploit-db.com, дата обращения 10.11.2008.
[20]. snapshot.debian.org. Режим доступа: http://snapshot.debian.org, дата обращения 10.11.2008.

Об одном подходе к анализу строк в языке Си для поиска переполнения буфера

И. А. Дудина <eupharina@ispras.ru>
Н. Е. Малышев <neket@ispras.ru>
Московский государственный университет имени М. В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1
Институт системного программирования им. В. П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Ошибки при работе с библиотечными функциями обработки строк в языке Си являются частой причиной переполнения буфера, что в свою очередь нередко приводит к отказу в обслуживании, некорректной работе программы или появлению эксплуатируемой уязвимости. Одним из способов устранения различных ошибок на стадии разработки программы является статический анализ. Существующие методы статического анализа, ориентированные на работу со строками, либо не обеспечивают должный уровень истинных срабатываний, либо пропускают большое количество ошибок, либо неприменимы к промышленным программам большого размера, либо реализованы в рамках закрытых инструментов. Для наиболее полного покрытия дефектов в реальных программах необходимо обнаруживать ошибки, происходящие лишь на некоторых путях выполнения и не определяемые единственной точкой программы, и, кроме того, находить ошибки, связанные с некорректным использованием не только библиотечных, но и пользовательских функций. Целью данного исследования является построение алгоритма поиска ошибок при работе со строками, удовлетворяющего этим свойствам, ограничению на количество ложных срабатываний (не более 40%), применимого к любым программам на языке Си и масштабирующегося на проекты из нескольких миллионов строк. Для решения этой задачи был использован ранее разработанный подход символьного исполнения с объединением состояний, который был адаптирован для поддержки строковых операций. На основе алгоритма отслеживания операций с целыми числами был предложен алгоритм отслеживания длин строк. Разработанный алгоритм реализован в качестве одного из детекторов семейства детекторов переполнения буфера в рамках инструмента статического анализа Svace. В результате на тестовом наборе Juliet test suite на тестах, связанных с переполнением правой границы буфера, покрытие срабатываниями увеличилось с 15,4% до 41,5%, при этом не было выдано ни одного ложного предупреждения. По сравнению с известным анализатором Infer на наборе Juliet инструмент Svace без поддержки строк показывает приблизительно те же результаты, за исключением случая сложных циклов, а связанные со строками переполнения Infer, как правило, не находит.

Ключевые слова: статический анализ; символьное исполнение; анализ строк

DOI: 10.15514/ISPRAS-2018-30(5)-3

Для цитирования: Дудина И. А., Малышев Н. Е. Об одном подходе к анализу строк в языке Си для поиска переполнения буфера. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 55-74. DOI: 10.15514/ISPRAS-2018-30(5)-3

1. Введение

Важным случаем ошибки переполнения буфера является переполнение при работе со строками. Строка в языке Си представляет собой массив символов, концом строки считается позиция ближайшего к началу нулевого элемента. Особенностью таких дефектов переполнения является тот факт, что работа со строками в языке Си преимущественно осуществляется с помощью специальных библиотечных функций. При этом, как правило, происходит доступ к элементам массива по различным индексам, наибольший из которых может быть равен длине строки. Это поведение само по себе небезопасно тогда, когда невозможно гарантировать, что длина строки заведомо меньше размера отведённого под неё массива. В таких случаях используют «безопасные» версии функций, дополнительно принимающие в качестве параметра число, с помощью которого ограничивается максимальный индекс доступа к строке, но даже такой подход не может гарантировать отсутствия ошибок.

Одним из способов устранения различных дефектов на стадии разработки программы является статический анализ. Существующие методы статического анализа, ориентированные на работу со строками, либо не обеспечивают должный уровень истинных срабатываний, либо пропускают большое количество ошибок, либо неприменимы к промышленным программам большого размера, либо реализованы в рамках закрытых инструментов. Для наиболее полного покрытия дефектов в реальных программах необходимо обнаруживать ошибки, происходящие лишь на некоторых путях выполнения и не определяемые единственной точкой программы, и, кроме того, находить ошибки, связанные с некорректным использованием не только библиотечных, но и пользовательских функций. Для выполнения этих требований анализ должен быть межпроцедурным и чувствительным к путям.

В данной работе предлагается подход к анализу строк в языке Си на основе символьного исполнения с объединением состояний. Рассматриваемый метод был реализован в рамках инструмента статического анализа Svace [10]. Дальнейшее изложение организовано следующим образом. В разд. 2 приводится краткое описание базового алгоритма, включающее описание внутрипроцедурного анализа методом символьного исполнения, межпроцедурного анализа с помощью метода резюме и общего подхода к поиску переполнения буфера. Разд. 3 посвящён расширению алгоритма символьного исполнения и алгоритма поиска буфера для анализа строк

в языке Си. Разд. 4 содержит результаты тестирования реализации рассмотренного метода. Описание существующих методов статического анализа для поиска ошибок при работе со строками приводится в разд. 5. Разд. 6 завершает статью.

2. Общий подход к поиску дефектов переполнения

2.1. Символьное исполнение с объединением состояний

Алгоритм внутрипроцедурного анализа основан на подходе символьного исполнения с объединением состояний. Анализ производится над разверткой графа потока управления на несколько итераций. Абстрактное состояние в каждой точке программы включает в себя предикат достижимости этой точки π и абстрактные значения переменных и ячеек памяти.

Абстрактные значения программы представляются символьными выражениями, множество которых обозначим как SE. К таковым относятся:

1) константные битовые вектора фиксированного размера;
2) символьные переменные (их множество обозначим $S \subset SE$);
3) арифметические операции над символьными выражениями.

Если множество переменных обозначить как V, то соответствие переменных их абстрактным значениям задаётся отображением $\sigma : V \rightarrow SE$. Предикат достижимости, в свою очередь, записывается как свободная от кванторов формула в теории битовых векторов, где в качестве переменных выступают символьные переменных из множества S.

В начальном состоянии на входе в функцию предикат достижимости тождественно равен истине; значениям формальных параметров и ячейкам в памяти сопоставлены различные новые символьные переменные. Анализ осуществляется путём продвижения абстрактного состояния по рёбрам графа развёртки.

При прохождении инструкции, изменяющей значение некоторой переменной или ячейки памяти, её символьное выражение в отображении σ обновляется в соответствии с семантикой инструкции. При прохождении через инструкции ветвления обновляется предикат достижимости: происходит конъюнкция предиката π с условием выбранной ветки, которое вычисляется с помощью σ. Если в некоторой инструкцию входит больше одного ребра на графе, то происходит слияние соответствующих абстрактных состояний на входе. При этом предикат достижимости вычисляется как дизъюнкция условий с входных рёбер, каждое из которых представляет собой конъюнкцию соответствующего предиката достижимости и условий равенства значения после объединения значению на рассматриваемой ветке для всех переменных и ячеек памяти.

2.2. Поиск переполнения буфера в рамках одной функции

Задачей анализа является обнаружение таких путей на графе развёртки (назовём эти пути *ошибочными*), которые, во-первых, являются выполнимыми и, во-вторых, прохождение которых всегда (при любых возможных значениях входных переменных) приводит к ошибке переполнения буфера. Такой подход выбран с целью, с одной стороны, обнаруживать дефекты, для которых не существует единственной ошибочной точки программы, и в то же время не выдавать большое количество ложных предупреждений, связанных с неизвестными возможными контрактами анализируемых функций [7].

Для решения этой задачи в абстрактное состояние программы было добавлено частичное отображение $VS : SE \rightarrow Summary$, которое для некоторых целочисленных значений программы, представленных символьными выражениями, определяет соответствующий элемент из множества *Summary*.

Элементы множества *Summary* обобщают информацию о некотором значении в данной точке программы, которая может быть использована для обнаружения ошибки переполнения, если это значение используется в качестве индекса для доступа к буферу. В произвольной точке программы q, если для некоторого символьного выражения $x \in SE$ известно $VS(x) = s \in Summary$, то для значения s и произвольного символьного выражения $h \in SE$, не зависящего от входных параметров, можно построить формулы $NotLess(q, s, h)$ и $NotGreater(q, s, h)$ в теории битовых векторов, удовлетворяющие следующему условию: для любого конкретного пути выполнения функции от начала функции до точки q, если выполнена формула $NotLess(q, s, h)$ ($NotGreater(q, s, h)$), то для пройденного пути по ГПУ при любых возможных для этого пути значениях входных параметров всегда в точке q выполнено $x \geq h$ ($x \leq h$). Таким образом, если в точке *ac* доступа к буферу размером S символьное выражения для индекса в данной точке равно $i \in SE$, предикат пути равен π и выполнима формула $\pi \wedge NotLess(ac, i, S)$, то существует путь на ГПУ, проходящий через точку *ac* и такой, что:

1. он является выполнимым (так как для некоторого набора входных переменных истинна формула π);
2. всегда на этом пути в точке *ac* выполнено $i \geq S$ (так как выполнена формула $NotLess(ac, i, S)$), то есть происходит переполнение буфера.

Заметим, что найденный путь удовлетворяет определению ошибочного пути, а, значит, выполнимость формулы $\pi \wedge NotLess(ac, i, S)$ является достаточным условием ошибки. Данный факт позволяет свести задачу поиска ошибок к задаче построения как можно более слабых и удовлетворяющих указанным выше требованиям условий $NotLess(q, s, h)$ и $NotGreater(q, s, h)$.

Подробное рассмотрение элементов множества *Summary* и построения искомых условий для них выполнено в статье [8]. Здесь в качестве примера

отметим, что одним из типов элементов этого множества являются константы, которые отображением *VS* всегда переводятся в себя. Для произвольной константы $c \in SE$ искомым условием является формула $NotLess(q, c, h) = c \geq h$. Действительно, так как значения c и h не зависят от входных переменных, то для любого пути ГПУ условие $c \geq h$ выполнимо либо для всех наборов значений входных параметров, либо ни для одного.

2.3. Межпроцедурный анализ

Межпроцедурный анализ производится с помощью метода резюме. Граф вызовов программы приводится к ациклическому виду разрывом обратных рёбер, и затем все функции программы анализируются единожды в обратном топологическом порядке. В результате внутрипроцедурного анализа функции формируется и сохраняется для последующего использования так называемое «резюме» — краткое описание поведения функции в терминах самого анализа. Далее при анализе инструкций вызовов пользовательских функций в силу порядка анализа функций гарантируется (в отсутствие рекурсии), что вызываемая функция уже проанализирована, значит, для неё уже имеется резюме, которое применяется в точке вызова.

Для простоты будем считать, что в резюме функции записывается объединённое абстрактное состояние из состояний в точках возврата из функции. Рассмотрим в общих чертах, как при вызове функции происходит применение её резюме, т.е. с помощью сохранённого абстрактного состояния вызванной функции модифицируется текущее состояние в точке вызова. Это происходит в несколько этапов:

1. Символьным переменным, соответствующим в резюме формальным аргументам функции, ставятся в соответствие символьные выражения, соответствующие в точке вызова фактическим аргументам функции. Переменной, в которую в точке вызова сохраняется результат функции, ставится в соответствие символьное выражение, соответствующее в резюме возвращаемому значению.
2. Обновляются отображение σ и предикат π в точке вызова. Символьные выражения «мигрируют» в контекст вызываемой функции рекурсивно, базой рекурсии является соответствие символьных переменных и символьных выражений, полученное в п.1.
3. При сопоставлении символьных выражений мигрируют также значения *VS* для этих выражений [8].

Для организации межпроцедурного поиска переполнения буфера в первую очередь необходимо поддержать случай межпроцедурного вычисления индекса. Для этого реализовано отслеживание зависимостей между целочисленными параметрами функции и возвращаемого и изменяемых внутри функции целочисленных значений (с учётом условий путей).

Кроме этого, также важно учесть возможность межпроцедурного доступа к буферу. Для инструкций доступа, корректность которых возможно проверить только во внешнем контексте (буфер и/или индекс вычисляются из параметров), в резюме функции добавляется условие ошибки. Условие при применении резюме мигрирует в контекст вызова, где либо проверяется (если информации уже достаточно), либо снова записывается в резюме, если проверить его по-прежнему можно только в вызываемой функции.

3. Анализ строк

Поддержка строковых операций будет заключаться, во-первых, в расширении абстрактного состояния отображением, задающим длину каждой строки, и учётом его значений при построении предиката π. Это позволит обнаруживать несовместные из-за ограничений на длины строк пути, что сократит количество ложных срабатываний не только для детектора переполнения буфера при работе со строками (для которого анализ выполнимости таких путей особенно критичен), но и для остальных чувствительных к путям детекторов.

Во-вторых, предлагается расширить отображение *VS* для обнаружения ошибок при работе со строками. Это позволит использовать уже имеющийся механизм анализа целочисленных значений для анализа длин строк.

3.1. Расширение абстрактного состояния для работы со строками

Содержимое каждой строки в абстрактном состоянии представлено одним символьным выражением, означающем длину данной строки. Выбор такой абстракции, с одной стороны, поможет найти больше ошибок и не выдавать ложных предупреждений на невыполнимых путях, предикаты которых содержат условия на длины строк. С другой стороны, добавление всего лишь одного символьного выражения для каждой строки не приведёт к значительному увеличению размера абстрактного состояния.

Таким образом, к абстрактному состоянию программы добавляется новое отображение $Slen : P \rightarrow SE$, где P — множество переменных, которые могут указывать на строку с точки зрения языка Си (например, можно выбрать множество всех переменных указательного типа).

В начальном состоянии программы каждой переменной из P сопоставляется новая символьная переменная. Строковым литералам сопоставляются константы, равные значению длин этих литералов в беззнаковом представлении.

Расширяются передаточные функции для инструкций присваивания переменных из множества P, арифметических операций над ними, инструкций доступа к массиву.

Также среди инструкций вызова функции отдельно рассматриваются инструкции вызова библиотечных функций работы со строками. Рассмотрим соответствующую передаточную функцию на примере функции `strncpy`.

$$\text{STRNCPY}\frac{\begin{array}{c} Slen_{in} \vdash \texttt{src} \to ls \quad Slen_{in} \vdash \texttt{dst} \to ld \quad \sigma \vdash n \to vn \\ \pi' = \bigvee \begin{array}{l} vn >_u ls \ \wedge \ lr = ls \\ vn \leqslant_u ls \ \wedge \ vn \leqslant_u ld \ \wedge \ lr = ld \\ vn \leqslant_u ls \ \wedge \ vn >_u ld \ \wedge \ lr \geqslant_u vn \end{array} \end{array}}{Slen_{out} = Slen_{in}\{\texttt{dst} \mapsto lr\} \quad \pi_{out} = \pi_{in} \wedge \pi'}$$

Данная функция принимает три параметра: адрес `dst`, куда копируются данные, адрес `src`, откуда копируется строка, и целочисленное значение `n`, определяющее наибольшее количество скопированных байт. Пусть на ребре, входящем в инструкцию вызова данной функции, значения отображения *Slen* для строк `dst` и `src` равны *ld* и *ls* соответственно, символьным выражением для переменной `n` является *vn*. Тогда на выходном ребре из этой инструкции для значения длины `dst` выберем новую символьную переменную *lr*, а условия на её значения добавим к предикату точки.

Для определения значения *lr* следует рассмотреть три случая. Если значение переменной `n` больше длины `src` (*vn* > *ls*), то длина `dst` после вызова будет равняться длине `src` (*lr* = *ls*). Если значение `n` не больше длины `src` (*vn* ≤ *ls*), то среди первых `n` байт строки `src` заведомо нет нулевого, и здесь возможны два случая. Если значение `n` также не превосходит длину `dst` (*vn* ≤ *ld*), то длина `dst` останется прежней (*lr* = *ld*), так как положение ближайшего к началу строки нулевого байта не изменится. В противном же случае про длину `dst` можно сказать лишь, что она заведомо не меньше `n`. Аналогичные построения можно провести и для других функций работы со строками.

Связь между значениями целочисленных переменных и длинами строк возникает при вызове функций, вычисляющих длину строки, и работе с массивами посимвольно. Так, например, при обработке инструкции `x = strlen(str)` символьное выражение для длины строки `str` копируется для переменной `x`.

$$\text{STRLEN}\frac{Slen_{in} \vdash \texttt{str} \to ls}{\sigma_{out} = \sigma_{in}\{x \mapsto ls\}}$$

При присваивании нового значения в элемент массива `str[i] = x` возможны два случая: если присваиваемое значение равно нулю и индекс меньше текущей длины, то длина строки `str` станет равна `i` либо не изменится в противном случае.

$$\text{BUFASSIGN}\frac{\begin{array}{c} Slen_{in} \vdash \texttt{str} \to ls \quad \sigma \vdash \texttt{x} = vx \quad \sigma \vdash \texttt{i} = vi \\ \pi' = \bigvee \begin{array}{l} (vx \neq 0 \ \vee \ ls <_u vi) \ \wedge \ lr = ls \\ (vx = 0 \ \wedge \ ls \geqslant_u vi) \ \wedge \ lr = vi \end{array} \end{array}}{Slen_{out} = Slen_{in}\{\texttt{str} \mapsto lr\} \quad \pi_{out} = \pi_{in} \wedge \pi'}$$

Поддержка нового отображения требуется и при объединении состояний в точках слияния: при объединении значений переменных из множества P объединяются не только их символьные выражения, но и длины их строк объединяются аналогичным способом с учётом условий на объединяемых ветках.

Для организации межпроцедурного анализа объединение отображений *Slen* для точек выхода из функции помещается в резюме функции. При применении резюме в точках вызова значения *Slen* из резюме транслируются в контекст вызывающей функции и сопоставляются соответствующим переменным из множества *P* по тому же принципу, как происходит трансляция отображения σ.

3.2. Расширение отображения *VS* для обнаружения ошибок при работе со строками

Идея рассматриваемого алгоритма заключается в том, чтобы анализировать длины строк аналогично тому, как анализируются целочисленные переменные, при этом обеспечивая совместимость этих двух анализов, то есть чтобы информация о длинах строк могла быть использована для анализа целочисленных значений и обратно. С этой целью отображение *VS* расширяется для символьных выражений, сопоставляемых строкам отображением *Slen.*

Как уже было сказано, все константы отображением *VS* переводятся в себя, поэтому для строковых литералов, значения *Slen* для которых являются константами, расширение *VS* происходит естественным образом.

Стандартные операции над строками моделируются по аналогии с операциями с целыми числами: так, вызов функции `strcpy` моделируется как присваивание длин, `strcat` — как сложение и т. п. Для проверки вызовов библиотечных функций на возможное переполнение строятся формулы *NotLess* и *NotGreater*, аналогичные тем, что используются при анализе инструкций доступа к буферу, но в качестве индекса используется максимальное смещение, по которому в соответствии с семантикой инструкции будет производиться доступ к строке.

Рассмотрим описанный в общих чертах подход на примере. В функции, приведённой на листинге 1, путь 2-3-4-5-6-7 является ошибочным. Чтобы обнаружить эту ошибку, необходимо доказать, что для некоторого пути в точке на строке 7 всегда `strlen(s)` $\geq$ `10` и `n` $>$ `10`.

```
1  void foo (int cond, int n) {
2      char s[100], dst[10]="";
3      if (cond)
4          strcpy(s, "very very long string");
5      if (n > 15)
6          //...
7      strncat(dst, s, n);
8  }
```

Листинг 1. Пример ошибки
Listing 1. Defect example

Первое условие заведомо выполнено, если в `s` была скопирована строка, длина которой всегда не меньше 10, т.е. если искомый путь проходит через точку 4 и `strlen("very very long string")>10`. Для точки 7 это можно записать как `cond ∧ 21>10`.

Второе условие выполнено, если было выполнено сравнение на строке 5 и `15≥10`. В точке 7 достаточным условием этого будет формула `n>15 ∧ 15≥10`.

С учётом того, что предикат достижимости точки 7 тождественно равен истине, итоговое достаточное условие ошибки будет иметь вид:

`cond ∧ 21>10 ∧ n>15 ∧ 15≥10.`

Данное условие построено таким образом, что наличие хотя бы одного удовлетворяющего ему набора значений входных переменных `n` и `cond` автоматически означает наличие ошибочного пути. Легко подобрать такие значения: `n = 16, cond = 42`. Подставив эти конкретные значения в условия переходов функции, можно получить искомый ошибочный путь.

4. Реализация и результаты

4.1. Реализация детектора

Задачей настоящего исследования было улучшение статического анализатора Svace [10] путём расширения чувствительного к путям анализа поддержкой строк языка Си. В предыдущей версии был реализован нечувствительный к путям анализ длин строк, основанный на интервальном анализе. Как следствие, информации, полученной в результате данного анализа, не было достаточно для организации чувствительного к путям поиска переполнения буфера при обработке строк. Кроме того, длины строк не учитывались для построения достаточных условий точки при символьном исполнении. Также имелся нечувствительный к путям детектор переполнения буфера при обработке строк, но он выдавал большое количество ложных срабатываний и не обеспечивал хорошее покрытие ошибочных ситуаций.

Для решения этих проблем была реализована поддержка строк при символьном исполнении с помощью алгоритма, описанного в разделе 3.1, и расширен имеющийся детектор переполнения буфера в соответствии с подходом, описанным в разделе 3.2. Рассмотренные алгоритмы были реализованы как для обычных строк, так и для строк с широкими символами (wide characters — символы, хранящиеся в типе `wchar_t` языка Си). Также был реализован нечувствительный к путям детектор ошибки, связанной с использованием обычной строки в качестве «широкой» строки, например при вызове функции `wcscpy`.

Для оценки масштабируемости новой версии Svace она была протестирована на проекте Android 5.0.2 без заметного ухудшения производительности по сравнению с базовой версией. Было выдано шесть истинных предупреждений, связанных с переполнением буфера при вызове обёртки над функцией `strcpy`.

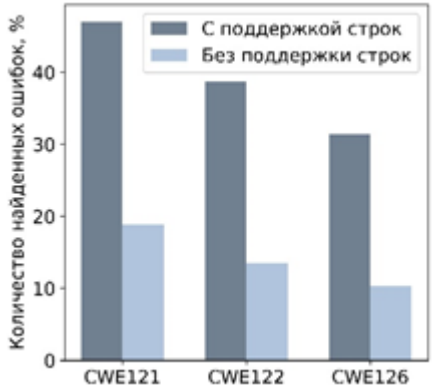

Рис. 1 Срабатывания анализатора на различных CWE
Fig. 1. Reported warnings by CWE

4.2. Тестирование с помощью Juliet Test Suite

Для тестирования использовался набор синтетических тестов Juliet Test Suite C/C++, разработанный в NSA's Center for Assured Software [11]. Для языков Си/Си++ в этом наборе представлен ряд тестов, размеченных по классификации CWE. К ошибке переполнения буфера среди всех групп набора относятся группы CWE 121 — «Stack-based Buffer Overflow» (переполнение буфера на стеке при записи), CWE 122 — «Heap-based Buffer Overflow» (переполнение буфера на куче при записи), CWE 124 — «Buffer Underwrite» (запись за левой границей буфера), CWE 126 — «Buffer Over-read» (чтение за правой границей буфера), CWE 127 — «Buffer Under-read» (чтение за левой границей буфера). Для задач настоящего исследования интерес представляли группы CWE 121, CWE 122, CWE 126, т.к. при работе со строками переполняется, как правило, правая граница массива.

Для каждого теста из набора также указан номер т. н. *варианта потока* (flow variant), который соответствует определенному виду потока управления и потока данных для данного теста. Среди вариантов потока управления рассматриваются различные случаи условий перехода, в том числе проверка глобальной переменной, условие из глобальной функции, различные виды оператора языка (`switch`, `while`, …). Варианты потока данных описывают различные случаи межпроцедурной и внутрипроцедурной передачи данных, такие как: пересылка через аргументы функции (в т. ч. по указателю, ссылке, как элемент массива или коллекции и пр.), через возвращаемое значение функции, через глобальную переменную. Некоторые из вариантов специфичны для языка Си++ и не применимы к тестам на Си.

Существует также классификация тестовых функций по *функциональным вариантам* (functional variants) — с учётом зависящих от конкретного типа дефекта особенностей тестового примера. В случае переполнения буфера к таким критериям можно отнести: имя библиотечной функции, вызов которой привёл к переполнению, тип элементов массива, способ выделения памяти под массив и т.п. Данная классификация, как правило, отражается в имени файла с тестом, например, `char_alloca_ncpy`.

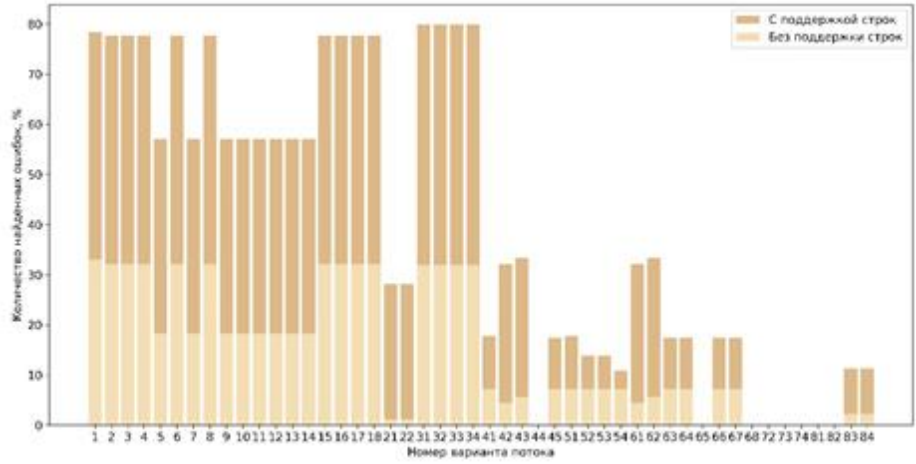

Рис. 2. Срабатывания анализатора на различных вариантах потока
Fig. 2. Reported warnings by flow variants

Изучение распределения тестов в интересных с точки зрения данного исследования группах показывает, что тесты распределены примерно равномерно между всеми вариантами потока. Почти треть тестов использует строки с широкими символами. Также значительное количество тестов проверяют использование библиотечных функций, таких как `memcpy`–подобные функции, функции обработки строк, использующие форматную

строку, и т. п. На рассмотренных группах тестов был дважды запущен анализатор Svace: с включенной поддержкой строк и без. Далее приведен анализ результатов тестирования.

Во-первых, ни на одном из запусков не было отмечено ложных срабатываний детекторов ошибки переполнения буфера на выбранных группах тестов.

Во-вторых, анализ результатов показал, что число срабатываний в интересующих группах увеличилось примерно в 2,5-3 раза (см. рис. 1). Общее число обнаруживаемых ошибок увеличилось с 15,4 % до 41,5 %. Также отдельно были рассмотрены функции, использующие тип `wchar_t`; число срабатываний детекторов на них увеличилось примерно в пять раз.

В-третьих, было произведено сравнение результатов внутри групп, соответствующих вариантам потока. Число срабатываний в 40 из 48 группах выросло в 2-10 раз (см. рис. 2). Оставшимся вариантам потока в обоих запусках соответствует нулевое количество срабатываний — эти варианты потока не поддерживаются (к таковым относятся, например, вызов функции по указателю, вызов виртуальной функции, передача данных через коллекции языка Си++).

Аналогичным образом рассматривались группы тестов, объединённые по функциональным вариантам. Результаты для 49 % от общего числа групп не изменились после реализации алгоритма — это в первую очередь тесты, в которых переполнение буфера не связано с работой со строками. При этом для 21 % групп было выдано нулевое количество срабатываний; в основном, это тесты, где используются функции с форматными строками (например, `snprintf`). Разбор форматной строки, необходимый для нахождения ошибок в этих тестах, реализован в инструменте Svace в отдельном детекторе, который не входит в рамки данной работы. Число срабатываний для остальных 51 % групп было нулевым до включения поддержки строк и стало равно 30-80 % от числа функций в группе. При этом лучшие результаты были получены на функциях, где ошибка возникает при использовании библиотечных функций (`strcat`, `strcpy`, `memcpy`, `memmove`), а худшие там, где ошибка возникает в цикле или при использовании функций с форматной строкой.

4.3. Сравнение с инструментом Infer

Для оценки эффективности метода было произведено сравнение со статическим анализатором Infer static analyzer [14, 15]. Этот инструмент разрабатывается компанией Facebook, имеет открытый исходный код и интенсивно развивается в настоящее время. Он активно используется в индустрии, например, в таких крупных IT-компаниях, как Amazon, Spotify, Uber, Mozilla Corporation [12].

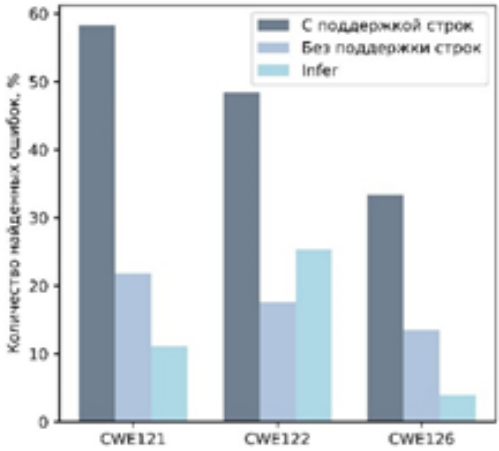

Рис. 3 Срабатывания анализаторов Svace и Infer на различных CWE
Fig. 3. Svace and Infer warnings by CWE

Для поиска ошибок переполнения буфера в Infer имеется экспериментальный детектор InferBO [13]. На данный момент для этого детектора заявлена только поддержка языка Си, поэтому из тестового набора, описанного выше, были исключены функции на языке Си++. На оставшихся функциях были протестированы версии Svace с и без поддержки строк в сравнении с анализатором Infer (см. рис. 3).

Так же, как и Svace, Infer не имеет ложных срабатываний на рассмотренном наборе. Процент обнаруживаемых ошибок составляет 15,2 % для Infer, 19,5 % для Svace без поддержки строк и 48,7 % при использовании описываемого метода. Говоря об отдельных группах CWE, можно отметить, что Infer лучше базовой версии Svace справляется с ошибками переполнения буфера на куче и хуже с ошибками переполнения буфера на стеке и чтения за границей буфера, значительно уступая версии с анализом строк во всех группах.

Только в трех вариантах потока из 37 ошибки не были обнаружены ни одним инструментом. Infer с различными вариантами потока справляется примерно одинаково, находя 15-20 % ошибок в 31 варианте и 0 % в оставшихся 7. В 15 вариантах базовая версия Svace превосходит Infer на 10-15 процентных пункта, на 1-2 пункта уступая в остальных вариантах с ненулевым числом найденных ошибок. В 23 вариантах версия Svace с поддержкой строк находит значительно больше ошибок чем Infer, на 30-60 процентных пунктов, выигрывая 0-10 процентных пунктов в остальных вариантах.

Табл. 1. Сравнение результатов Svace и Infer
Table 1. Warnings types detected by Svace and Infer

	Число функций в группе	**Svace с поддержкой строк, %**	**Svace без поддержки строк, %**	**Infer, %**
`Memcpy`	1 294	63,1	33,9	23,8
`Memmove`	1 258	64,5	34,9	24,5
`Loop`	1 222	22,9	16,5	23,7
`Cpy/Cat/Ncat`	1 532	65,7	0,0	0,0
`Ncpy`	652	56,7	15,0	17,8
`Other`	1 350	20,1	18,4	6,7

При сравнении результатов по вариантам ошибки в целом наблюдается следующая картина. Практически все функциональные варианты, покрываемые Infer, обнаруживает и базовая версия Svace. При этом на большинстве этих вариантов Infer показывает на 10-15 процентных пунктов лучший результат, в основном для тех функций, которые имеют дополнительную пометку CWE 805 – «Buffer Access with Incorrect Length Value» (доступ к буферу с некорректным значением длины). В то же время Svace обнаруживает несколько функциональных вариантов, которые Infer не обнаруживает вовсе, за счет чего имеет немного более высокий общий процент обнаруживаемых ошибок. Версия с поддержкой строк обнаруживает более чем в 2 раза больше различных функциональных вариантов по сравнению с Infer.

В табл. 1 приведены результаты инструментов в укрупнённых группах функциональных вариантов. Данные группы были получены путем объединения функциональных вариантов по последнему слову их имени. Например, функциональный вариант `char_type_overrun_memcpy` попадет в группу `memcpy`. Это было сделано для краткости в связи с очень большим количеством функциональных вариантов. В группе `loop` ошибка проявляется во время обращений к буферу по индексу, вычисляемому в цикле и превышающему его размер. В группах `memcpy`, `memmove` ошибка возникает при вызове соответствующих библиотечных функций с некорректным параметром размера буфера, в группе `Cpy/Cat/Ncat` — при вызове функций `strcpy`, `strcat`, `strncat`. Отдельно была рассмотрена группа `ncpy` с библиотечной функцией `strncpy`, для обнаружения переполнения первого аргумента которой не нужно знать длины строк-аргументов. Функциональные варианты, не попавшие ни в одну из перечисленных групп, были объединены в группу `other`. Они содержат небольшое количество функций и характеризуются относительно низким количеством срабатываний каждого анализатора.

Из таблицы видно, что Infer лучше обрабатывает циклы и чуть лучше, чем Svace без поддержки строк, находит ошибки, связанные с `strncpy`, при этом показывая более слабый результат на функциях `memmove` и `memcpy`. Также можно заключить, что поддержка строк в Infer отсутствует.

Следует отметить, что тестовый набор Juliet не является репрезентативной выборкой типов дефектов, встречающихся в коде реальных программ, поэтому на основании только лишь результатов тестирования нельзя сделать исчерпывающих выводов о качестве и сравнительной практической пользе рассматриваемых инструментов.

5. Обзор существующих решений

Одним из первых статических анализаторов, ориентированных на анализ строк для поиска переполнения, стал созданный в 2000 г. инструмент BOON [1]. Анализатор моделирует каждую строку парой переменных, определяющих размер выделенной памяти и длину строки. В процессе обхода абстрактного синтаксического дерева исходной программы строится система целочисленных интервальных неравенств. После решения полученной системы для каждой строки анализируется условие безопасности: если не доказано, что длина строки меньше размера буфера, то выдается предупреждение. Анализ является межпроцедурным, нечувствительным к потоку и контексту вызова. Исследования независимых авторов [2, 3] отмечают хорошую производительность, но при этом большое количество ложных срабатываний. Количество обнаруживаемых ошибок также существенно ниже по сравнению с другими детекторами переполнения буфера, что отчасти может быть объяснено узкой специализацией инструмента как анализатора строковых операций.

В 2003 г. был создан инструмент CSSV [4], целью которого является обнаружение всех ошибок переполнения буфера в программе на языке CoreC (подмножество Си) с небольшим количеством ложных срабатываний. Анализ осуществляется отдельно для каждой функции, межпроцедурный анализ организован с помощью аннотаций, предоставленных пользователем. CSSV преобразует исходную программу в программу над целочисленными значениями и консервативно проверяет её на наличие ошибок. Необходимость написания аннотаций пользователем существенно ограничивает применимость данного инструмента. К недостаткам также можно отнести отсутствие масштабируемости на большие программы.

Подход, предложенный в работе [5], как и CSSV, гарантирует обнаружение всех ошибок переполнения при работе со строками. Он основан на абстрактной интерпретации с использованием полиэдров в качестве абстракции для моделирования возможных значений размеров массивов и длин строк. В работе [6] предлагается в качестве более точной абстракции использовать *тропические* полиэдры — аналог выпуклых полиэдров в тропической алгебре. Такой подход позволяет вычислять более точные

инварианты над длинами строк в результате строковых преобразований. Данные методы позволяют гарантировать обнаружение всех ошибок рассматриваемого типа, однако плохо масштабируются на программы большого размера.

Ряд современных промышленных анализаторов, таких как Coverity Prevent, Klocwork, HP Fortify, включают в себя детекторы переполнения буфера при работе со строками, однако используемые ими алгоритмы закрыты, что затрудняет суждения о качестве этих детекторов.

Перспективными в контексте анализа строк для переполнения буфера представляются решатели с поддержкой строк [9]. В настоящее время они получили широкое применение для анализа значений строк в веб-приложениях, где они позволяют обнаруживать и автоматически анализировать процедуры проверки входных данных («санитайзеры») для устранения потенциальных уязвимостей. Способность данных инструментов анализировать не только размер, но и содержимое строк может быть также полезна и при анализе переполнения буфера, однако число реально встретившихся нам на практике примеров, для которых требуется выносить суждения о содержимом строк, относительно невелико.

6. Заключение

В статье представлен метод поиска переполнений буфера, происходящих при работе со строками языка Си. Метод основан на символьном исполнении с объединением состояний, является межпроцедурным, обладает чувствительностью к путям выполнения и контекстам вызовов. Идея метода заключается в расширении для операций со строками предложенного в предыдущих работах подхода [7, 8], который заключается в отслеживании операций с целочисленными значениями и построению на этой основе достаточных условий возникновения ошибки.

Разработанный метод реализован в статическом анализаторе Svace и обладает теми же показателями масштабируемости и точности анализа, что и основные детекторы ошибки переполнения буфера. В частности, для тестов из пакета Juliet поиск переполнений строк позволил резко увеличить покрытие (число находимых ошибок) в части, относящейся к переполнению буфера,— рост составил около трёх раз. Оставшаяся не найденной часть ошибок в основном связана с недостатками не предложенного метода как такового, а основной части (ядра) анализатора Svace — это анализ сложных циклов и коллекций языка Си++. Работы над улучшением анализатора в этих направлениях ведутся в настоящее время.

Сравнение результатов тестирования на пакете Juliet с результатами статического анализатора Infer позволил сделать вывод, что в Infer поддержка строк при поиске переполнений буфера не реализована, а обычные переполнения ищутся примерно с тем же качеством, что и в базовой версии Svace. Infer лучше обрабатывает тестовый исходный код со сложными

циклами, что подтверждает наш вывод о необходимости доработки Svace в этом направлении.

С точки зрения доработки предложенного метода перспективным направлением авторам видится интеграция решателей с поддержкой строк (Z3str2, Z3str3, CVC4 и т.п.), что позволит находить ошибки переполнения, для которых нужно выносить суждения о содержимом строк.

Список литературы

[1]. Wagner D., Foster J., Brewer E., Aiken A. A first step towards automated detection of buffer overrun vulnerabilities. In Proc. of the Network and Distributed System Security Symposium, 2000, pp. 3-17.

[2]. Zitser M., Lippmann R., Leek T. Testing static analysis tools using exploitable buffer overflows from open source code. ACM SIGSOFT Software Engineering Notes, vol. 29, issue 6, 2004, pp. 97-106.

[3]. Kratkiewicz K. Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code. Master's Thesis, Harvard University, 2005.

[4]. Dor N., Rodeh M., Sagiv M. CSSV: Towards a Realistic Tool for Statically Detecting All Buffer Overflows in C. ACM SIGPLAN Notes, vol. 38, 2003, pp. 155-167.

[5]. Simon A., King A. Analyzing String Buffers in C. Algebraic Methodology and Software Technology, vol. 2422, 2002, pp. 365-379.

[6]. Allamigeon X. Static analysis of memory manipulations by abstract interpretation. Algorithmics of tropical polyhedra, and application to abstract interpretation. PhD Thesis, École Polytechnique, 2009.

[7]. Дудина И. А., Кошелев В. К., Бородин А. Е. Поиск ошибок доступа к буферу в программах на языке C/C++. Труды ИСП РАН, том 28, вып. 4, 2016 г., стр. 149–168, 2016. DOI: 10.15514/ISPRAS-2016-28(4)-9

[8]. Dudina I. A., Belevantsev A. A. Using static symbolic execution to detect buffer overflows. Programming and Computer Software, vol. 43, issue 5, pp. 277–288, 2017. DOI: 10.1134/S0361768817050024

[9]. Zheng, Y., Ganesh, V., Subramanian, S., Tripp, O., Berzish, M., Dolby, J., Zhang, X. Z3str2: an efficient solver for strings, regular expressions, and length constraints. Formal Methods in System Design, vol. 50, 2017, pp.249-288.

[10]. А.Е. Бородин, А.А. Белеванцев. Статический анализатор Svace как коллекция анализаторов разных уровней сложности. Труды ИСП РАН, том 27, вып. 6, pp. 111—134, 2015. DOI: 10.15514/ISPRAS-2015-27(6)-8

[11]. Juliet Test Suite v1.2 for C/C++. User Guide. Center for Assured Software National Security Agency, December 2012.

[12]. Infer static analyzer Infer. URL: https://fbinfer.com/ (Дата обращения: 21.09.2018)

[13]. Inferbo: Infer-based buffer overrun analyzer. URL: https://research.fb.com/inferbo-infer-based-buffer-overrun-analyzer/ (Дата обращения: 21.09.2018)

[14]. Calcagno C., Distefano D. et al. Moving Fast with Software Verification. Lecture Notes in Computer Science, vol. 9058, 2015, pp. 3-11.

[15]. Calcagno C., Distefano D., O'Hearn P., Hongseok Y. Compositional Shape Analysis by means of Bi-Abduction. In Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL '09), 2009, pp. 289-300.

An approach to the C string analysis for buffer overflow detection

I. A. Dudina <eupharina@ispras.ru>
N. E. Malyshev <neket@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. Many buffer overrun errors in C programs are caused by erroneous string manipulations. These can lead to denial of service, incorrect computations or even exploitable vulnerabilities. One approach to eliminate such defects in the course of program development is static analysis. Existing static analysis methods for analyzing strings either produce many false positives, miss too many errors, scale poorly, or are implemented as a part of a proprietary software. To cover a significant amount of the real program defects it is necessary to detect errors that could happen only on a particular program path and cannot be defined by a single erroneous point. Also, it is essential to find misusage of library functions and user functions. The aim of this study is to develop a detection algorithm that will cover such cases, will produce at most 40% false warnings, will be applicable to any C programs without any additional restrictions, and will scale up to millions of lines of code. We have extended our approach of symbolic execution with state merging to support string manipulations. Also we have developed a string overflow detector based on our buffer overflow approach with integer indices. The new algorithm was implemented in the Svace static analyzer. As a result, the coverage of the buffer-overflow related testcases from the Juliet test suite has increased from 15.4% to 41.5% with zero false positives. Also we have compared our Juliet results to those of the Infer static analyzer. The basic Svace version (without string support) is on par with Infer except for the flow variant of complex loops, whereas string-related buffer overflows are not detected by Infer.

Keywords: static analysis; static symbolic execution; string analysis

DOI: 10.15514/ISPRAS-2018-30(5)-3

For citation: Dudina I. A., Malyshev N. E. An approach to the C string analysis for buffer overflow detection. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 55-74 (in Russian). DOI: 10.15514/ISPRAS-2018-30(5)-3

References

[1]. Wagner D., Foster J., Brewer E., Aiken A. A first step towards automated detection of buffer overrun vulnerabilities. In Proc. of the Network and Distributed System Security Symposium, 2000, pp. 3-17.

[2]. Zitser M., Lippmann R., Leek T. Testing static analysis tools using exploitable buffer overflows from open source code. ACM SIGSOFT Software Engineering Notes, vol. 29, issue 6, 2004, pp. 97-106.

[3]. Kratkiewicz K. Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code. Master's Thesis, Harvard University, 2005.

[4]. Dor N., Rodeh M., Sagiv M. CSSV: Towards a Realistic Tool for Statically Detecting All Buffer Overflows in C. ACM SIGPLAN Notes, vol. 38, 2003, pp. 155-167.
[5]. Simon A., King A. Analyzing String Buffers in C. Algebraic Methodology and Software Technology, vol. 2422, 2002, pp. 365-379.
[6]. Allamigeon X. Static analysis of memory manipulations by abstract interpretation. Algorithmics of tropical polyhedra, and application to abstract interpretation. PhD Thesis, École Polytechnique, 2009.
[7]. Dudina I. A., Koshelev V. K., Borodin A. E. Statically detecting buffer overflows in C/C++. Trudy ISP RAN/Proc. ISP RAS, vol 28, issue 4, pp. 149–168, 2016. DOI: 10.15514/ISPRAS-2016-28(4)-9
[8]. Dudina I. A., Belevantsev A. A. Using static symbolic execution to detect buffer overflows. Programming and Computer Software, vol. 43, issue 5, pp. 277–288, 2017. DOI: 10.1134/S0361768817050024
[9]. Zheng, Y., Ganesh, V., Subramanian, S., Tripp, O., Berzish, M., Dolby, J., Zhang, X. Z3str2: an efficient solver for strings, regular expressions, and length constraints. Formal Methods in System Design, vol. 50, 2017, pp.249-288.
[10]. Borodin A., Belevantcev A. A static analysis tool Svace as a collection of analyzers with various complexity levels. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 6, 2015, pp. 111—134.
[11]. Juliet Test Suite v1.2 for C/C++. User Guide. Center for Assured Software National Security Agency, December 2012.
[12]. Infer static analyzer Infer. URL: https://fbinfer.com/ (Дата обращения: 21.09.2018)
[13]. Inferbo: Infer-based buffer overrun analyzer. URL: https://research.fb.com/inferbo-infer-based-buffer-overrun-analyzer/ (Дата обращения: 21.09.2018)
[14]. Calcagno C., Distefano D. et al. Moving Fast with Software Verification. Lecture Notes in Computer Science, vol. 9058, 2015, pp. 3-11.
[15]. Calcagno C., Distefano D., O'Hearn P., Hongseok Y. Compositional Shape Analysis by means of Bi-Abduction. In Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL '09), 2009, pp. 289-300.

Approach to analyzing executable code based on the software architecture recovery

D.S. Kononov <dspr2@yandex.ru>
Federal State Unitary Enterprise «18 CSRI», Ministry of Defence of RF,
4 Svobodny prospect, Moscow, Russia, 111123

Abstract. The article discusses a new approach to obtaining additional information about the software module under study based on the preliminary software architecture recovery during the executable code analysis. As a result, it is possible to reduce the requirements for the resources spent by limiting the field of research, rational choice of priorities, and abstraction from secondary elements. The paper demonstrates the feasibility of restoring the software architecture in a two-step process: first, the separate components are isolated, and then their purposes and relationships are determined. An automated method for decomposing a software module is proposed, which allows allocating components corresponding to static libraries, classes, and their groups. This method is based on the functions clustering by the distances between them in the address space and on the call graph. A description of the implementation of the developed method as a plug-in for the IDA disassembler is given.

Keywords: executable code analysis; software architecture; clustering; call graph; distance between functions; software module; decomposition

DOI: 10.15514/ISPRAS-2018-30(5)-4

For citation: Kononov D.S. Approach to analyzing executable code based on the software architecture recovery. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 75-88. DOI: 10.15514/ISPRAS-2018-30(5)-4

1. Introduction

The need to analyze the executable code is widely encountered in practice when addressing issues related to the protection of intellectual property, the search for software backdoors and vulnerabilities, the analysis of computer viruses, certification of compilers, and software development. It should be noted that despite all the achievements in this area, the task in question is still far from being solved. This is due to the fundamental limitations expressed in the extreme complexity of formalizing and automating the analysis of executable code (Каушан, Маркин, Падарян, & Тихонов, 2013). If, when searching for vulnerabilities, there are still effective automatic methods (fuzzing, symbolic execution), then, when restoring the executable code algorithm, the result of the study is largely determined by the quality of expert analysis. In this case, automation is limited to local signature or statistical methods that facilitate the search for constants or a special set of

instructions in specific algorithms. Particularly acute shortcomings of the modern scientific and methodological apparatus appear when it is impossible to use dynamic analysis methods. Taking into account the volume of the software modules under study, which in the case of embedded systems reach tens and hundreds of thousands of functions, the solution of practical tasks in this area requires significant investments of both material and time resources and the availability of substantial human capital.

Thus, it becomes necessary to provide an expert with information about a software module, important for achieving a positive result. As practice shows, first of all, an expert needs to understand the logic of the software module (Quist & Liebrock, 2009), since it allows targeted search through "reverse engineering" (Streekmann, 2012, стр. 27)p. 27]. In this case, the model of the expert's work changes: instead of the direct restoration of the algorithms implemented in the program code, the assumptions regarding their implementations are confirmed and specific parameters are determined.

One of the important components of the "reverse engineering" approach is the restoration of the software architecture, information about which is lost during compilation. Software architecture (Streekmann, 2012, стр. 9)p. 9] can be described in the framework of a hierarchical model of the structure of complex systems (Косяков & Свит, 2014).

2. Hierarchical model of software

The development of complex software requires high-quality software architecture with well-defined systems and subsystems that solve a specific problem and have weak coupling (Макконнелл, 2010, стр. 96). Modern non-specialized programming languages clearly support such programming paradigm (Microsoft Corp., 2006). When examining a software module, it is possible to break it up into components that correspond to the initially programmed systems and subsystems (for example, using source code or debugging information (Ebert, Riediger, & Winter, 2008), (Bohnet & Dollner, 2006)). For definiteness, one can designate such selected subsets of a software module as components, and the process of breaking up a software module – decomposition. According to the model used, the set of components is hierarchical, in which the components located at the upper level consist of a combination of the underlying ones. As part of the research, the components classification is proposed, presented in Table 1.

Table 1. Classification of components

No.	Name	Description
1	Software	A component that fully incorporates the software under investigation. Always present in a single instance.
2	File	A separate file in the corresponding file system included in the software. For embedded systems without a file system, all firmware is considered as a single file.

3	Embedded operating system	For systems with multiple processors, there can be several embedded operating systems running in parallel within the same address space.
4	Static library	For software modules (including embedded operating systems) the presence of built-in libraries is typical. For example, OpenSSL encryption libraries, various drivers, libraries of standard functions (memcpy, strlen), etc. Components of a similar size (500-3000 functions) that have a weak connection (for example, only using API) with the rest of the code also belong to this level.
5	Class group	This level corresponds to a group of classes in the object-oriented programming terminology. Typical size from 100 to 1000 functions. Differ from level 4 in a greater connectivity with the rest of the code. For example, various classes that implement the same interface (plug-ins), network protocol handler, etc.
6	Class	This level corresponds to a class in the object-oriented programming terminology or an object (compiled) module in the C language. They have a size of up to 100 functions. Differ from level 5 in a greater connectivity with the rest of the code. For example, the implementation of a circular buffer, hash tables, etc.
7	Function	Currently, the task of the allocation of automating functions from executable code is solved and implemented in modern disassemblers at a sufficient level.
8	Logical block in a function	The part of the function consisting of basic units designed to solve a subtask. For example, inline functions, condition checking, loops, etc.
9	Basic unit	A sequence of instructions without transitions automatically isolated by modern disassemblers (cycle body, the condition being checked, etc.).
10	Logically isolated sequence of instructions	Part of the basic unit, designed to solve some subtasks. For example, loading data from memory, untwisted cycle, etc.
11	Instruction	Executable machine instruction. Automatically isolated by modern disassemblers.
12	Instruction argument	The executable machine instruction argument. Automatically isolated by modern disassemblers.

In the modern scientific and methodological apparatus for analyzing executable code, the information only about levels 1, 2, 3, 7, 9, 11, 12 is used due to the absence of additional debugging data (Meng & Miller, 2016). The existing significant gap between levels 3 and 7 makes it necessary to analyze software modules consisting of tens and hundreds of thousands of functions using methods

that have exponential computational complexity (fuzzing, character execution, etc.) (Каушан, Маркин, Падарян, & Тихонов, 2013). To overcome the existing limitations, it is necessary to take into account levels 4, 5, 6 (Streekmann, 2012), which will significantly reduce the requirement for the resources needed to conduct the software module investigation (Quist & Liebrock, 2009). Thus, existing methods with high computational complexity can be scaled for software modules with a volume of more than 10,000 functions due to their decomposition into components with a characteristic size falling within the range of effective application of the corresponding methods. For embedded systems, this process is, in fact, analogous to isolating dynamic libraries and programs from the general-purpose operating system (OS) and examining them separately. At the same time, to implement the considered approach, it is necessary that the separated integral parts have a specific isolated functionality, that is, they would be components according to the terminology adopted in the paper.

There are a large number of software architecture definitions (Clements, Bachmann, & Bass, 2010, стр. 27), (Ian, 2011, стр. 20). The conceptual apparatus of the research is based on the IEEE 1471 standard (ANSI/IEEE Standart 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems, p. 9), p. 9]: software architecture is the fundamental organization of the system, embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. Thus, from the definition, it follows that the restoration of the disassembled software module architecture (Streekmann, 2012, стр. 30)p. 30] should be carried out in two stages. The first is the decomposition of the disassembled software module into levels 3-6 components, and the second is the determination of the functionality of the selected components and the restoration of their relationships with each other and with the environment.

3. Analysis of the executable code, taking into account the software architecture

In the modern scientific and methodological apparatus, the main element of the research is the function, which leads to the need to analyze and restore the algorithms of a large number of interrelated functions to determine their common purpose. In contrast, the preliminary decomposition of a software module allows determining the role of a specific component in the architecture by analyzing only a few of its functions (in some cases just one) or the data and strings used in it. As a result, based on the described approach, a reasoned conclusion is made on the assignment of hundreds and thousands of functions that form the corresponding component by analyzing a small amount of data. Furthermore, additional information about the purpose of the components is provided by an analysis of their relationships.

Knowledge of the software architecture makes it possible to rationally prioritize the research within the framework of solving a specific practical problem. For example,

if it is necessary to restore the network interaction protocol of the botnet, then the emphasis in the study should be placed on the appropriate handler. One of the features of this approach to restoring the executable code algorithm is the ability to limit the study of functions from non-priority components to the conceptual level. As an example, one can cite the situation when the algorithm of the bootloader is investigated and the component of interaction with flash memory is isolated. In this case, one should not restore the entire algorithm for writing or reading flash memory, but logically assign the values "write" or "read" to the component functions called from the bootloader. In addition, the joint analysis of a single software module by several experts is significantly simplified due to the rational differentiation of the studied areas into separate components.

Information about the software module architecture is also required when conducting dynamic analysis. Thus, the lack of information about the components used significantly complicates the analysis of execution routes and slicing (Падарян, Гетьман, & д.р., 2014). Even when examining programs for the Windows OS family under the x86 architecture, it is difficult to draw an unequivocal conclusion about the algorithm being performed and its purpose without separating the called functions by the system API. In the case of embedded OS («firmware»), this problem is only aggravated.

In the framework of «fuzzing», it is impossible to correctly emphasize its direction without knowing the architecture of the software module. The cases of work only within one component from the study of their entirety are indistinguishable. The availability of information about the software architecture makes it possible to rationally select the area of study, excluding components that are not interesting in the current context. This leads to the possibility of multiple reductions in computational costs. For example, by isolating the component of working with strings or with memory, one can prevent loop traversal in the functions of copying memory or comparing strings, replacing them with appropriate heuristics.

Despite all the advantages described, in modern scientific and methodological apparatus for analyzing executable code in the absence of debugging data, there are no effective automated methods not only for restoring the software architecture but also for decomposing a software module into components. As a result, such an approach is not used in practice in the overwhelming majority of cases, since the time and cost of resources do not pay off in the current realities. This situation significantly limits the ability to analyze executable code.

4. Method of the software module decomposition

As the applied methods for decomposing a software module into components, apart from the expert one, one can single out various modifications of the task of finding *strongly connected components* (Новиков, 2009, стр. 283) of the call graph and various imaging techniques. However, due to limited disassembling capabilities (Meng & Miller, 2016), low call graph density, and the presence of widely used functions (for example, working with strings and memory) that are called from

almost anywhere, there are low informative results that do not allow the software module decomposition (Streekmann, 2012, стр. 52, 96). Dynamic analysis methods simplify this process insignificantly (Quist & Liebrock, 2009), but they themselves are not applicable in the general case and work for relatively small amounts of code due to the coverage problem (Каушан, Маркин, Падарян, & Тихонов, 2013), (Падарян, Гетьман, & д.р., 2014). In this regard, similar approaches to the analysis of executable code are practically not used, although they are quite widespread for the source code (Bohnet & Dollner, 2006), (Kienle & Muller, 2010).

At the same time, the experience of software research has shown that often interrelated functions are located nearby in the address space. This arrangement is explained both by the optimization for the hardware architecture and by the simplification of compiler development. For example, with paging memory, a speed gain occurs when finding jointly called functions within a single page (Eagle, 2011, стр. 114). On the other hand, the simplest implementation of the linker involves the sequential addition of object modules (Bryant & O'Hallaron, 2016, стр. 672), and shuffling the functions between them implies some optimization. As a rule, an object module corresponds to a separate source code file (a class in object-oriented programming) and, therefore, is a component by definition in software with a well-developed architecture. Thus, to carry out the decomposition of the software module, it is proposed to perform clustering of functions based on distances both in the address space and on the call graph. It should be noted, however, that the interrelated functions are absolutely not obliged to be located near each other in the address space, but these cases are associated either with a significant level of optimization or with the use of some protection measures (for example, small granular randomization of address space allocation during compilation (Нурмухаметов, Курмангалеев, Каушан, & С.С., 2014)).

As the distance between two functions in the address space, it is proposed to use the number of positions enclosed between them in the list of functions sorted by starting address. Such a choice is explained by the need to eliminate the dependence of the distance on the size of the functions and the data placement order. However, it should be taken into account that there is a certain selected size of a component of a certain level (for example, 1000-3000 functions in the case of static libraries), and, at the same time, it is necessary to consider the interaction of all functions in the software module under study. Based on these prerequisites, in order to obtain the final distance in the address space, an increasing step function was chosen corresponding to the estimated sizes of the components at various levels (see Table 1).

$$d(f_i, f_j) = l(j - i)$$
$$l(x) = \begin{cases} 0, x = r_0 = 0 \\ k_1, 0 < x < r_1 \\ k_2, r_1 \le x < r_2 \\ \dots \\ k_K, r_{K-1} \le x < \infty \end{cases} \quad (1)$$
$$k_{i+1} > k_i; r_{i+1} > r_i$$
$$k_i, r_i \in \mathbb{N}$$

where f_i is a function with a sequence number i in the list of functions, sorted in order of increasing starting addresses; k_i – step function coefficient for the i-th range; r_i – limit of the i-th range; K – the number of ranges in the step function.

Then each edge of the call graph is assigned a weight equal to the distance $d(f_i, f_j)$ between the functions in the address space (1). As a result, the distance between functions on the call graph is defined as the minimum sum of edge weights that form the path from one function to another. It is necessary to clarify that the call graph, in this case, is considered as an undirected graph, that is, there is a path in the graph from the calling function to the called one, and vice versa. It should also be borne in mind that with an arbitrary choice of the step function, it is possible that the distance calculated from the call graph will be less than the corresponding value of the step function. The simplest example of this kind is

$$l(x) = \begin{cases} 0, x = 0 \\ 1, x = 1 \\ +\infty, x > 1 \end{cases} \quad (2)$$

To eliminate this inconsistency, it is necessary to ensure that the selected step function satisfies the condition: the coefficient value for each interval of the step function must be less than or equal to the sum of the coefficient values for the previous interval and the minimum possible edge weight (i.e., the coefficient of the first interval). Indeed, consider the first point of any interval and draw an edge to it from the previous point (located in the previous interval, respectively). Then if the difference between adjacent intervals is greater than the minimum coefficient of the step function, then the length of the edge from the origin to the selected point will be greater than the length of the path through the immediately preceding point. The formally described condition can be expressed by the formula (3).

$$k_{i+1} \le k_i + k_1 \quad (3)$$

Since the weight of the edge in this problem is non-negative, it is possible to use the Dijkstra algorithm to find the distances between all the functions (Кормен, Лейзерсон, Ривест, & Штайн, 2013, стр. 595)p. 595]. Then the computational complexity of finding the distances from the current function to all the others will be $O(n^2 + m)$, where n is the number of nodes (functions) on the call graph, and m is the number of edges (calls). Given that clustering requires the calculation of the distance matrix between all functions, then the total computational complexity will be $O(n^3 + nm)$. Using the binary heap in Dijkstra's algorithm can reduce the

computational complexity to $O(n^2 \log n + nm \log n)$. At the same time, it is necessary to consider that $m = O(n^2)$. However, in practice, the call graphs of real programs are strongly sparse: the density of call graphs for all checked software modules with volumes from a few hundred to tens of thousands of functions tends to 0. The latter is explained by the decrease in connectivity between subsystems with increasing software scale. The calculated values of the ratio m/n (the average number of edges per node) for the studied software modules did not exceed 4 and tends to decrease with an increase in the module volume. Consequently, in the analysis of the executable code, the relationship for the call graph is satisfied $m = O(n)$ and the evaluation can be used for the computational complexity of constructing the distance matrix $O(n^2 \log n)$.

In the framework of the experiments, it was found that the parameters of the step function can be specified considering the expected sizes of the components within fairly wide limits. Thus, the proposed method for decomposing a software module is robust. Additionally, this conclusion is confirmed by the fact that the experiments were conducted under the conditions of the existing limitations of modern means of disassembling (IDA software) to restore the call graph. As a result, one can conclude that the information about the original software architecture is stored in the executable code and can be restored.

5. Practical implementation

Currently, interactive clustering is implemented based on the creation of a heat map for the distance matrix. In the distance matrix, functions are sorted in ascending order of their starting addresses. In this case, no additional computational costs are required apart from converting the calculated matrix into a graphic image in the BMP format. An example of such an image ("code map") is shown in Fig. 1; the darker the point, the smaller the distance between the functions.

Interactive clustering is performed by manually selecting rectangular blocks visually different from adjacent areas. As a result, a hierarchical structure is formed on the "code map" consisting of a number of square blocks located on the diagonal, which either do not intersect or are nested in another block. This structure corresponds to the software architecture of the module under study, and the diagonal blocks themselves corresponds directly with the components of different levels. Blocks outside the diagonal determine the degree of interaction between the components. Also, for additional confirmation of the decision on the correctness of the components selection and the initial assessment of their assignment, strings and other data, which are used in the functions from the block selected on the "code map", are automatically displayed on request for the operator.

The following optimizations are added during implementation:

1) individual disconnected components of the call graph are excluded if the number of nodes is less than the threshold (the recommended value is 20);

2) individual disconnected call graph components having a diameter (Новиков, 2009, стр. 249)p. 249] less than the threshold are excluded (the recommended value of 3, excluding graphs with the star topology);
3) excluded functions that are called only from a single function and do not call anything;
4) springboard functions (mediating long-distance calls) are excluded from the distance matrix by signature, but are taken into account when constructing routes;
5) stub functions and imported functions are excluded.

The program for calculating the distance matrix and interactive clustering is implemented as a plug-in for the IDA disassembler. The minimum input data is the call graph and start addresses of functions, which allows analyzing the executable modules for unsupported IDA processors upon independent receipt of the specified data. Information about the selected components is stored in the IDB file and is used to display functions in the form of a tree-like list, similar to that used when displaying projects in modern integrated software development environments.

The PC with average computing capabilities was used as a test bench: dual-core processor with a frequency of 3.1 GHz (Core i3 2100), 8 GB RAM, SSD drive. In the study of software modules of up to 10,000 functions, the calculation and construction of the "code map" takes place within a minute. Such delays are insignificant in the context of the study of the program code for the operator. The applied step distance function is given by the formula 4:

$$l(x) = \begin{cases} 0, x = 0 \\ 1, 0 < x < 100 \\ 2, 100 \le x < 400 \\ 3, 400 \le x < 800 \\ 4, 800 \le x < 1600 \\ 5, 1600 \le x \end{cases} \quad (4)$$

The experiments performed using the example of Nmap software version 7.10 x86 for Windows OS[1] (Nmap.org, 2016) showed that the selected components in the executable code correspond to specific subsystems and classes in the source code (fig. 1). In addition, the dependence of the isolation degree of the components on their level and on the size of the software module was confirmed, which is fully consistent with the need to improve the quality of the software architecture while increasing the size and complexity of the project. In turn, the high quality of the latter is provided mainly by strengthening the cohesion of the components and weakening the coupling between them.

It should be noted that in the process of decomposition, the specific features of the software module (including those introduced by the compiler) are revealed, the information on which allows simplifying and automating the study of the executable

[1] nmap.exe module disassembled in IDA Pro 7.0 environment contains 3436 functions, 3082 functions were allocated after the use of heuristics.

code. Due to the additional analysis, functions of the main cycles, standard service functions of working with memory and strings, error handling functions, springboard functions, designed to link the high-level components to each other are determined.

6. Areas for further research

For the full implementation of the approach to analyzing the executable code considered in the article, it is necessary to develop automatic methods for restoring the software architecture, making it possible to explore the entire existing range of sizes of software modules. In addition, these methods should be universal both in terms of hardware architecture and the technologies, languages, and programming paradigms used. To achieve these properties, it is required to work out the issues of the restoration of components interconnections and effective hierarchical clustering, including the case of random allocation of address space.

In the near future, the proposed method for decomposing a software module will be developed. It is planned to implement automatic clustering methods; to take into account the relationship graph of functions based on the using data in addition to the call graph; to perform software modules classification based on the characteristics that affect the decomposition process (hardware architecture, programming paradigms, code size, etc.); to optimize the parameters of the step function of the distance for the extracted classes of modules.

It should be noted that the result is influenced by the quality of the call graph recovery and, accordingly, the improvement of this indicator is also one of the directions of the described approach development.

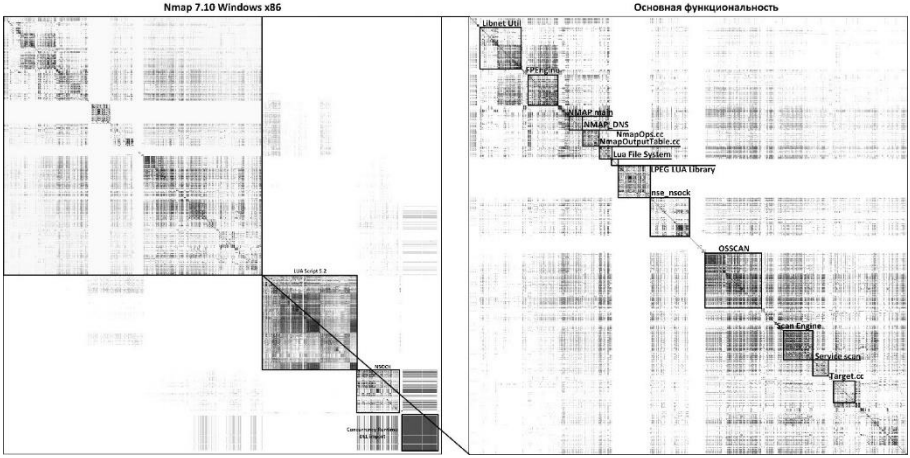

Fig. 1 – Code map of the Nmap.exe software module version 7.10 x86 for Windows OS after contrast correction. The components corresponding to the classes and their groups in the source code are partially labeled. Small parts are not displayed due to scale limitations

7. Conclusion

Knowledge of the software architecture makes it possible to significantly reduce the requirements for consumed resources during the analysis of executable code by limiting the field of research, rational choice of priorities, abstraction from secondary elements, and joint analysis. As a result, the software module under study is divided into separate components with a characteristic size of several thousand functions, which, in fact, leads to a decrease in the dimension of the original problem. Moreover, there is an additional way of expansion of the obtained intermediate results of the analysis for the entire software module. To achieve the indicated advantages, it is necessary to restore the software architecture of the executable code.

It is proposed to carry out this process in two stages: 1) decomposition of the disassembled software module into separate components; 2) the definition of the functionality of the selected components and their relationships. To perform the first stage, an automated method has been developed that allows selecting components corresponding to static libraries, classes, and their groups. This method is based on the functions clustering by the distances between them in the address space and on the call graph. Currently, interactive heat map clustering for the distance matrix is implemented as a plug-in for the IDA disassembler. The conducted experiments confirmed the possibility of restoring the software architecture only by the software module, which made it possible to demonstrate in practice the advantages of the approach to the analysis of executable code considered in the article.

References

[1]. Kaushan V.V., Markin Yu.V., Padaryan V.A., Tikhonov A.Yu. Methods for Finding Errors in a Binary Code. Technical Report. ISP RAS, Moscow, 2013 (in Russian).

[2]. Quist D.A., Liebrock L.M. Visualizing Compiled Executables for Malware Analysis. Proc. of the International Workshop on Visualization for Cyber Security (VisSec09), 2009, pp. 27-32.

[3]. Streekmann N. Clustering-Based Support for Software Architecture Restructuring. Springer, 2012, 241 p.

[4]. Kosyakov A., Svit U. Systems Engineering. Principles and Practice. 2nd ed. Moscow: DMK Press, 2014, 624 p. (in Russian).

[5]. McConnell S. Code Complete. Workshop. 2nd ed. Moscow: Russian edition, 2010, 896 p. (in Russian).

[6]. Microsoft Corp. ECMA-334 C# Language Specification. Ecma International. 2006. Available at: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf, accessed 13.10.2017.

[7]. Ebert J., Riediger V., Winter A. Graph Technology in Reverse Engineering. The TGraph Approach, Proc. of the 10th Workshop Sowtware Reengineering (WSR 2008), vol. 126, 2008, pp. 67-81.

[8]. Meng X., Miller B.P. Binary Code Is Not Easy. Proc. of the 25th International Symposium on Software Testing and Analysis (ISSTA16), Saarbrucken, Germany, 2016, pp. 24-35.
[9]. Clements P., Bachmann F., Bass L. et al. Documenting Software Architectures: Views and Beyond. 2nd ed. Addison-Wesley Professional, 2010, 517 p.
[10]. Ian G. Essential Software Architecture, 2nd ed. Springer, 2011, 242 p.
[11]. ANSI/IEEE Standard 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems.
[12]. Padaryan V.A., Getman A.I. et al. Methods and Software Supporting the Combined Analysis of a Binary Code. Programming and Computer Software, vol. 40, issue 5, 2014, pp. 276–287.
[13]. Novikov F.A. Discrete Mathematics for Programmers: Textbook for Universities, 3rd ed. Piter, 2009, 384 p. (in Russian).
[14]. Bohnet J., Dollner J. Visual Exploration of Function Call Graphs for Feature Location in Complex Software Systems. Proc. of the 2006 ACM Symposium on Software Visualization, 2006, pp. 95-104.
[15]. Kienle H.M., Muller H.A. Rigi – An Environment for Software Reverse Engineering, Exploration, Visualization and Redocumentation. Science of Computer Programming, vol. 75, issue 4, 2010, pp. 247-263.
[16]. Eagle C. IDA Pro Book, 2nd ed. No Starch Press, 2011, 672 p.
[17]. Nurmukhametov A.R., Zhabotinsky E.A., Kurmangaleev S.F., Gaisaryan S.S., Vishnyakov A.V. Fine-grained address space layout randomization on program load. . Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 163-182 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-9
[18]. Bryant R.E., O'Hallaron D.R. Computer Systems: A Programmer's Perspective. 3rd ed. Pearson, 2016, 1084 p.
[19]. Nurmukhametov A.R., Kurmangaleev S.F., Kaushan V.V., Gaisaryan S.S. Compiler protection techniques against software vulnerabilities exploitation. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 3, 2014, pp. 113-126 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-6
[20]. Kormen T. Kh., Leyzerson Ch.I., Rivest R.L., Stein K. Algorithms: Construction and Analysis, 3rd ed. Williams LLC, 2013, 1328 p. (in Russian).
[21]. Nmap: the Network Mapper, 2016. Available at: https://nmap.org/dist/nmap-7.10-win32.zip, accessed 21.08.2018.

Подход к анализу исполняемого кода на основе восстановления программной архитектуры

Д.С. Кононов <dspr2@yandex.ru>
ФГУП «18 ЦНИИ» МО РФ,
111123, Россия, г. Москва, Свободный проспект, д. 4.

Аннотация. В статье рассматриваются новый подход к получению дополнительной информации об исследуемом программном модуле на основе предварительного восстановления программной архитектуры в ходе анализа исполняемого кода. В результате появляется возможность сократить требования к затрачиваемым ресурсам за счёт ограничения области исследования, рационального выбора приоритетов, абстрагирования от второстепенных элементов. В работе демонстрируется

осуществимость восстановления программной архитектуры в рамках двухэтапного процесса: вначале проводится выделение обособленных компонентов, а затем определяются их назначения и взаимоотношения. Предлагается автоматизированный метод декомпозиции программного модуля, позволяющий выделять компоненты, соответствующие статическим библиотекам, классам и их группам. Данный метод базируется на кластеризации функций по расстояниям между ними в адресном пространстве и на графе вызовов. Приведено описание реализации разработанного метода в виде плагина для дизассемблера IDA.

Ключевые слова: анализ исполняемого кода; программная архитектура; кластеризация; граф вызовов; расстояние между функциями; программный модуль; декомпозиция.

DOI: 10.15514/ISPRAS-2018-30(5)-4

Для цитирования: Кононов Д.С. Подход к анализу исполняемого кода на основе восстановления программной архитектуры. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 75-88 (на английском языке).. DOI: 10.15514/ISPRAS-2018-30(5)-4

Список литературы

[1]. Каушан В.В., Маркин Ю.В., Падарян В.А., Тихонов А.Ю. Методы поиска ошибок в бинарном коде, технический отчет, ИСП РАН, Москва, 2013.
[2]. Quist D.A., Liebrock L.M. Visualizing Compiled Executables for Malware Analysis. Proc. of the International Workshop on Visualization for Cyber Security (VisSec09), 2009, pp. 27-32.
[3]. Streekmann N. Clustering-Based Support for Software Architecture Restructuring. Springer, 2012, 241 p.
[4]. Косяков А., Свит У. Системная инженерия. Принципы и практика. 2-е изд. Москва: ДМК Пресс, 2014. 624 с.
[5]. Макконнелл С. Совершенный код. Мастер-класс. 2-е изд. Москва: Издательство «Русская редакция», 2010. 896 с.
[6]. Microsoft Corp. ECMA-334 C# Language Specification. Ecma International. 2006. URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf (дата обращения: 13.Октябрь.2017).
[7]. Ebert J., Riediger V., Winter A. Graph Technology in Reverse Engineering. The TGraph Approach, Proc. of the 10th Workshop Sowtware Reengineering (WSR 2008), vol. 126, 2008, pp. 67-81.
[8]. Meng X., Miller B.P. Binary Code Is Not Easy. Proc. of the 25th International Symposium on Software Testing and Analysis (ISSTA16), Saarbrucken, Germany, 2016, pp. 24-35.
[9]. Clements P., Bachmann F., Bass L. et al. Documenting Software Architectures: Views and Beyond. 2nd ed. Addison-Wesley Professional, 2010, 517 p.
[10]. Ian G. Essential Software Architecture, 2nd ed. Springer, 2011, 242 p.
[11]. ANSI/IEEE Standard 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems.
[12]. Падарян В.А., Гетьман А.И. и др. Методы и программные средства поддерживающие комбинированный анализ бинарного кода. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 251-276. DOI: 10.15514/ISPRAS-2014-26(1)-8

[13]. Новиков Ф.А. Дискретная математика для программистов: Учебник для вузов. 3-е изд. СПб: Питер, 2009. 384 с.
[14]. Bohnet J., Dollner J. Visual Exploration of Function Call Graphs for Feature Location in Complex Software Systems. Proc. of the 2006 ACM Symposium on Software Visualization, 2006, pp. 95-104.
[15]. Kienle H.M., Muller H.A. Rigi – An Environment for Software Reverse Engineering, Exploration, Visualization and Redocumentation. Science of Computer Programming, vol. 75, issue 4, 2010, pp. 247-263.
[16]. Eagle C. IDA Pro Book, 2nd ed. No Starch Press, 2011, 672 p.
[17]. Нурмухаметов А.Р., Жаботинский Е.А., Курмангалеев Ш.Ф., Гайсарян С.С., Вишняков А.В. Мелкогранулярная рандомизация адресного пространства программы при запуске. Труды ИСП РАН. том 29, вып. 6, стр. 163-182. DOI: 10.15514/ISPRAS-2017-29(6)-9
[18]. Bryant R.E., O'Hallaron D.R. Computer Systems: A Programmer's Perspective. 3rd ed. Pearson, 2016, 1084 p.
[19]. Нурмухаметов А.Р., Курмангалеев Ш.Ф., Каушан В.В., С.С. Г. Применение компиляторных преобразований для противодействия эксплуатации уязвимостей программного обеспечения. Труды ИСП РАН, том 26, вып. 3, стр. 113-126. DOI: 10.15514/ISPRAS-2014-26(3)-6
[20]. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ. 3-е изд. Москва: ООО «И.Д. Вильямс», 2013. 1328 с.
[21]. Nmap.org. Nmap: the Network Mapper 2016. URL: https://nmap.org/dist/nmap-7.10-win32.zip (дата обращения: 21.08.2018).

Platform for interprocedural static analysis of binary code

H.K. Aslanyan <hayk@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper describes the developed platform for static analysis of binary code. The platform is developed based on interprocedural, flow-sensitive and context-sensitive analysis of the program. The machine-independent language REIL is used as an intermediate representation. In this representation basic data flow analyzes are developed and implemented - reaching definitions analysis, construction of DEF-USE and USE-DEF chains, analysis for deletion of dead code, value analysis, taint analysis, memory analysis and etc. The implemented approach for functions' annotations allow propagating data between function calls, thereby making the context-sensitive analysis. The platform provides an API for using all implemented analyzes, which allows adding new analyzes as plugins.

Keywords: static analysis; binary code analysis; interprocedural analysis

DOI: 10.15514/ISPRAS-2018-30(5)-5

For citation: Aslanyan H.K. Plarform for interprocedural static analysis of binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 89-100. DOI: 10.15514/ISPRAS-2018-30(5)-5

1. Introduction

Software developers always strive to create high-quality software, meaning that it should be reliable, safe and easy to maintain. However, with increasing size and complexity of projects, the developed code contains more errors [1]. Fixing those errors can be done at any phase of the software development life cycle. Ideally, all errors are detected during the testing phase. Error detection at the later phases or after deployment may cause many difficulties. Moreover, erroneous software may result in money loss. However, even a very thoroughly tested software sometimes contains errors. Currently, various code analysis tools are widely used to detect these errors.

Static code analysis is one of the common defect detection approaches. Static analysis examines examining a code without executing a program. Through a complete analysis of syntax, semantics, control and data flow, static code analysis can find errors that are difficult or impossible to find during testing, especially on

rarely executed paths. Static analysis is based on methods and approaches both from fundamental and applied research.

There are lots of approaches for static analysis of source code [2–8]. However, static analysis of executables is less studied, despite the fact that it has several advantages over the source code analysis. The first advantage of the binary code analysis compared to the source code analysis is the fact that the source code is not always available. The second advantage is that aggressive compiler optimizations may create defects in binary code that were non-existent in the source, and it is very hard to prove the optimization correctness [9-10]. The third advantage is the undefined semantics of certain language constructs that may create difficulties for a static analyzer. For example, in C/C ++ the order in which actual function parameters are evaluated is implementation defined, which can lead to false positive reports in the source code analyzer.

A production quality static analysis tool should have the following features: interprocedural analysis support, flow sensitivity, path sensitivity. In addition, a high-quality analyzer should be able to analyze large files (binary file sizes can reach hundreds of megabytes) in a few hours, provide high accuracy (a small number of false positives), and it should be easy to extend for supporting new error types.

2. Platform architecture

The proposed tool architecture was developed taking into account the following requirements:

- target architecture independent;
- context-sensitive interprocedural analysis with flow-sensitive intraprocedural analysis;
- scalability: analyzing tens of megabytes of executable files in a few hours;
- easy platform extension.

The first step is producing assembler code from an executable. Assembler language instructions are created by a disassembler using the object code as input. The tool uses the IDA Pro [2] disassembler since it supports many executable file formats for a large number of processors, automatically restores control flow graphs and call graphs. The disassembler also restores calling conventions. Then the resulting assembly code is transferred to the Binnavi [3] tool, which converts it to the REIL representation (Reverse Engineering Intermediate Language) [4]. REIL representation is an intermediate low-level language that can be used to write platform-independent analysis algorithms. It has only 17 instructions. Each instruction calculates no more than one result and has no side effects (flag settings, etc.). REIL representation is created for a virtual processor with unlimited memory and an unlimited number of registers denoted as t0, t1, t2, etc. Target machine registers can be also accessed in REIL. Fig. 1 shows the scheme for getting the assembler and REIL representation.

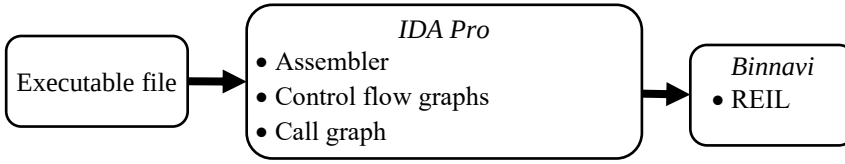

Fig. 1.Getting a REIL representation

3. Function summaries and interprocedural analysis

After generating a REIL representation, the call graph is made acyclic. First, the classical Tarjan approach [5] is used to find strongly connected components (SCCs). Second, directed cycles are identified, and an arbitrary edge is removed from each of them. This process breaks the connectivity properties of the SCCs.

Then call graph nodes are divided into groups (fig. 2) as follows: the first group has nodes that have no outgoing edges. The second group includes nodes whose descendants are in the first group. Thus, each subsequent group includes the nodes that have their successor nodes processed as belonging to the previous groups. Since the call graph has no more directed cycles, the algorithm will be completed in a finite number of steps, and each node will fall into a certain group.

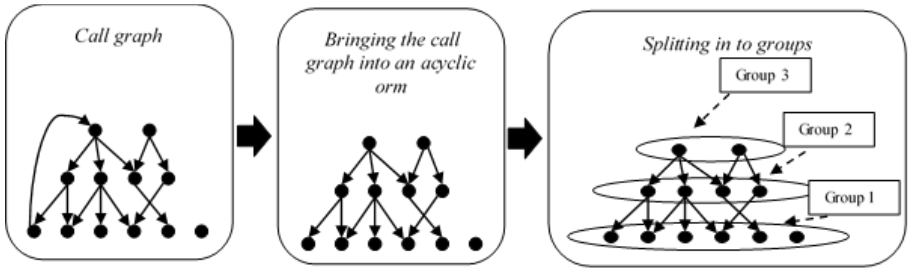

Fig. 2. Splitting nodes of the call graph into groups

Next, call graph traversal is performed according to the node groups built. Intraprocedural analyses are run starting from the first group's nodes, and each next group is only considered if the functions corresponding to all previous group's nodes have been analyzed. It should be noted that the analysis is performed only for functions with available bodies, i.e. functions from dynamic libraries are not analyzed (only summaries are available for such functions). When interprocedural analyses are completed, so-called function *summaries* are saved (summaries contain function-specific data calculated by the analyses). For example, a function returns the value that is user-controlled (like e.g. `gets` function in C/C++), or a function frees the memory pointed to by the first parameter. When analyzing a function, its callees' summaries are used. Obviously, in the absence of recursive calls, all called functions' summaries are available. In the case of recursive calls, some edges are

removed from the call graph and thus some called function summary may not be available. Such cases are handled as calls of unknown functions (without a summary). We have used the C standard library summaries from the Svace tool [6]. Also, summaries can be extended with new types of data in our platform.

Intraprocedural analyses for each function are run only once, which allows achieving scalability w.r.t. the number of functions. Splitting call graph nodes into groups gives the advantage of analyzing the functions within each group in parallel.

4. Intraprocedural analysis

Basic intraprocedural analyses that form the platform contents are performed using the REIL representation. Function summaries are used when processing function calls, and the analysis data is evaluated taking into account actual call parameters and calling conventions. This process makes the analysis context-sensitive. Currently, value analysis, reaching definitions analysis, DEF-USE and USE-DEF chains construction, dead code removal, liveness analysis, taint analysis, and dynamic memory analysis are implemented. The intraprocedural analysis architecture makes it easy to extend the set of analyses (fig. 3) and to add plugins.

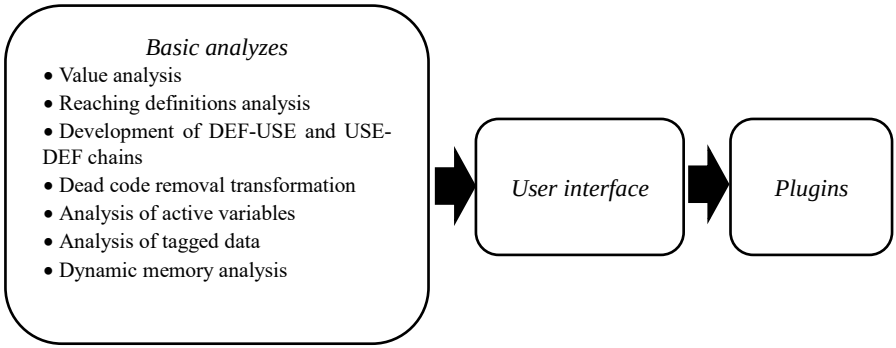

Fig. 3. Intraprocedural analysis architecture

2.1 Value analysis

Value analysis is used to track values in registers and memory cells. All registers (target architecture registers and temporaries) and all memory cells that are used in the program are called *variables*. During the analysis process all variables get values for all program points. For values stored in memory, a memory model, which tracks memory accesses for stack, heap, and static memory areas, was developed and implemented.

Value analysis is implemented based on a classic iterative data flow approach [7]. For this purpose, a semilattice is defined, that is, initial values for all variables are

specified and transfer functions are defined. All other analyses are based on the value analysis.

4.1.1 Value types

The developed value analysis has several symbolic value types: an integer type, a target architecture register, a temporary REIL register, a memory area, and a special values bottom and top. The bottom value is assigned to variables that have unknown value (the lowest element in the semilattice), and the top value is assigned to variables that may have any value (the uppermost element in the semilattice). Fig. 4 shows the value analysis semilattice.

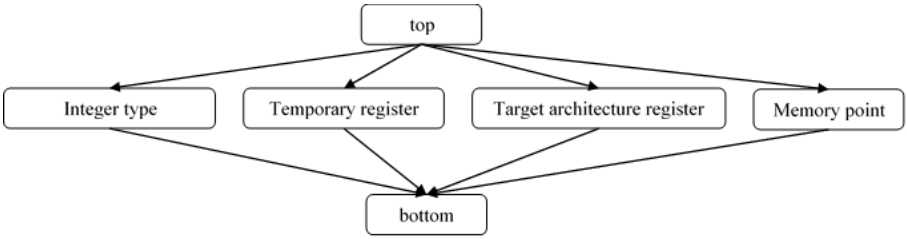

Fig. 4. Diagram of the value analysis semilattice

4.1.2 Memory model

A simple memory model is just a byte array. Memory stores and loads in this model are emulated as stores or loads to the corresponding array element. However, such a simple model has some drawbacks. It is impossible to determine concrete addresses for certain memory areas, e.g. those that are heap allocated. Moreover, function calls sequence may change during each subsequent program run, which will generally result in the ambiguity of memory values.

For proper analysis, the tool must separate different memory areas. To address the challenge, the following memory model is proposed. Memory is addressed as follows: `*(reg + constants_array) + constant`, where `reg` is a register, `constants_array` is an array of constant values, and `constant` is a constant value. `constants_array` and `constant` play the role of offset, and `constants_array` provides the ability to model multidimensional array elements and structure fiels. `reg` has a basic symbolic value. It is important to note that all formula elements are not necessarily needed to model the given cell.

- **Stack memory model.** Since it is impossible to determine the precise value of the function stack top statically, the model refers to local variables by the offset from the stack top of the current function. Therefore, the symbol `stack` for the initial address of the analyzed function's stack is used, and all local variables are modeled relative to this address. For example, in the x86 architecture, after the instruction `mov eax, esp+4` the value of `eax` will be `stack+4`, and after the instruction `mov ebx,`

`[esp+8]` the value of ebx will be `*(stack+{8})`. `constants_array` provides the ability to model values of structure fields. For example, if the value of the `a->b->c` expression in C code is in `ebx`, then after processing all REIL instructions the value of `constants_array` for `ebx` will be `{offset b in structure a, offset c in structure b}`.

- **Heap memory model.** To model heap memory accesses, the `heap` symbol is used and the instruction address of the memory allocation function call is put into `constants_array`. For example, after processing the `malloc` call with the address equal to `0xFFFFFFFF` (on the x86 architecture with the cdecl calling convention), `eax` will be `*(heap+{0xFFFFFFFF})`.
- **Static memory model.** Static and global variables are modeled directly with their address with or without an offset. After compilation, all static variables' addresses are known, and the variable address is put in `constants_array`, and its offset is put to `constant`.

4.1.3 Value analysis implementation

The value analysis algorithm is based on the iterative data flow approach [7]. The semilattice, transfer functions and initial values are defined. The top/bottom semilattice elements are denoted as `top`/`bottom`, respectively. `Bottom` is the initial value for all variables except stack top and function arguments.

Transfer functions are defined for REIL instructions as follows. Let us define the register value `ti` as `Val(ti)`. For example, for the instruction `add t1, t2, t3` (it adds the value `t1` to `t2` and stores in `t3`) the transfer function is defined as follows: all variables' values remain unchanged except for `t3`, and `Val(t3)` will be defined as follows:

- `top`, if `Val(t1)=top` or `Val(t2)=top;`
- `bottom`, `if Val(t1)=bottom` or `Val(t2)=bottom;`
- `Val(t1)+Val(t2)`, if `Val(t1)` and `Val(t2)` are integer constants;
- `*(reg+constants_array)+(constant+v)`, if `Val(t1)=*(reg + constants_array)+constant` and `Val (t2)` is an integer constant that is equal to `v;`
- `*(reg+constants_array)+(constant+v)`, if `Val(t2)=*(reg + constants_array)+constant` and `Val (t1)` is an integer constant that is equal to `v;`
- `*(reg)+v`, if `Val(t1)` is a register that equals to `reg`, and `Val (t2)` is an integer constant that is equal to `v;`
- `*(reg)+v`, if `Val(t2)` is a register that equals to `reg` and `Val (t1)` is an integer constant that is equal to `v;`
- `top`, if none of the above applies.

Similarly, transfer functions for other 17 REIL instructions are defined. The iterative algorithm converges as we limit the number of calculated values for each variable, so the algorithm stops when no values are changed or the above limit is reached.

4.2 Data flow analyses implementation

Based on the value analysis, other classical data flow analyses are implemented (reaching definitions analysis, dead code removal, liveness analysis, taint analysis, and dynamic memory analysis). The above analyses are also performed using the iterative data flow algorithm [7]. Semilattices and transfer functions are similarly defined, and initial values are assigned to variables. DEF-USE and USE-DEF chain construction is based on reaching definitions. The platform provides an API for working with all existing analyses, which allows implementing new analyses as plugins.

5. Related work

Balakrishnan and Reps describe in [8] an approach for analyzing value intervals. It is implemented in the CodeSurfer/x86 tool, which can be used to analyze executables for the x86 architecture. The tool uses the IDA Pro disassembler to restore the program assembly code, its control flow graphs and the call graph. The tool implements a memory model, based on which the interprocedural, context-sensitive value interval analysis is performed.

In [9] [10] [11], platforms for analyzing x86 executables are developed and implemented. These works implement an intermediate language and a disassembler, also adapting value interval analysis of [8] values for their intermediate representation. In [10], tainted data analysis is developed in addition to the above.

The paper [12] presents the BAP tool for analyzing executable files built for the x86 and ARM architectures. Both dead code removal and DEF/USE chain construction are implemented, but the analyses do not take into account memory data dependencies, which significantly lowers their quality.

The platform described in our work has two main functional advantages: it does not depend on target architecture and uses the function summary approach, which allows achieving linear scalability w.r.t. the number of analyzed functions.

6. Experimental results

All algorithms described in the paper were implemented and tested on real and artificial examples. Table 1 shows running times of all the described analyses for lepton, php and clam projects. Tests were run on a machine with a Core i5 processor, 4 cores and 16 GB RAM.

As can be seen from the table, php has a larger size compared to clam, but the analysis time of this project is shorter. Such results can be explained by the fact that

functions in the clam project are much larger on average than php functions. Therefore, parallel function analysis in php is much more efficient.

Table 1. Experimental results

Executable file	Architecture	Size	The time of all analyses
lepton	x86	5 MB	19 min 21sec
php	x64	29 MB	3 h 12min
clam	x86	18 MB	4 h 20min

7. Conclusion and further work

In this work, we have presented a platform for binary code analysis that is target independent and supports a variety of classical data flow analyses. The application of the developed platform using the implemented APIs can be found in [20-24]. These projects, in particular, used reaching definitions analysis and USE-DEF/DEF-USE chains for building program dependency graphs.

In the future, we plan to add analyzers for finding critical errors in binary code. In addition, as the REIL representation does not support floating point numbers, the described analyses currently work only with integer types, and we plan to add such support, which will increase the analyzers' accuracy.

References

[1]. S. C. Misra and V. C. Bhavsar. Relationships between selected software measures and latent bug-density: Guidelines for improving quality. In Proc. of the International Conference on Computational Science and its Applications, ICCSA, Monreal, Canada, 2003.

[2]. V. P. Ivannikov, A. A. Belevantsev, A. E. Borodin, V. N. Ignatiev, D. M. Zhurikhin and A. I. Avetisyan. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, no. 5, 2014, pp. 265-275.

[3]. Coverity scan. Synopsys, https://scan.coverity.com/.

[4]. Klocwork static code analysis. RogueWave software, https://www.roguewave.com/products-services/klocwork/static-code-analysis.

[5]. Fortify Static Code Analyzer. Micro Focus, https://software.microfocus.com/ru-ru/products/static-code-analysis-sast/overview.

[6]. IBM AppScan. IBM, https://www.ibm.com/us-en/marketplace/ibm-appscan-source.

[7]. V. K. Koshelev, V. N. Ignatiev, A. I. Borzilov and A. A. Belevantsev. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, no. 4, 2017, pp. 268–276.

[8]. A. A. Belevantsev. Multilevel static analysis for improving program quality. Programming and Computer Software, 2017, pp. 321–336.

[9]. G. Balakrishnan and T. Reps. WYSINWYX: What You See Is Not What You eXecute. ACM Transactions on Programming Languages and Systems, vol. 32, no. 6, 2010, pp. 1-84.

[10]. H. J. Boehm. Threads cannot be implemented as a library. In Proc. of the 2005 ACM SIGPLAN conference on Programming Language Design and Implementation, 2005, pp. 261-268.
[11]. IDA Pro disassembler. Hex-Rays, https://www.hex-rays.com/products/ida.
[12]. Binnavi. Zynamics, https://www.zynamics.com/binnavi.html.
[13]. REIL - The Reverse Engineering Intermediate Language. Zynamics, https://www.zynamics.com/binnavi/manual/html/reil_language.htm.
[14]. R. E. Tarjan. Depth-first search and linear graph algorithms. In Proc. of the 12th Annual Symposium on Switching and Automata Theory, 1971, pp. 114 - 121
[15]. V. Aho, R. Sethi and J. D. Ullman. A formal approach to code optimization. In Proceedings of a Symposium on Compiler Optimization, 1970, pp. 86-100.
[16]. J. Kinder. Static analysis of x86 executables. Ph.D. Thesis, Technische Universitat Darmstadt, 2010.
[17]. S. Cheng, J. Yang, J. Wang, J. Wang and F. Jiang. LoongChecker: Practical Summary-Based Semi-simulation to Detect Vulnerability in Binary Code. *In* Proc. of the 10th International Conference on Trust, Security and Privacy in Computing and Communications, Changsha, 2011, pp. 150-159.
[18]. D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam and P. Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. In Proc. of the 4th International Conference on Information Systems Security, 2008, pp. 1-25.
[19]. D. Brumley , I. Jager , T. Avgerinos and E. J. Schwartz. BAP: A Binary Analysis Platform. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 463-469.
[20]. H. K. Aslanyan. Effective and Accurate Binary Clone Detection. Mathematical Problems of Computer Science, vol. 48, 2017, pp. 64-73.
[21]. G. S. Keropyan, V. G. Vardanyan, H. K. Aslanyan, S. F. Kurmangaleev and S. S. Gaissaryan. Multiplatform Use-After-Free and Double-Free Detection in Binaries. Mathematical Problems of Computer Science, vol. 48, 2017, pp. 50-56.
[22]. H. Aslanyan, A. Avetisyan, M. Arutunian, G. Keropyan, S. Kurmangaleev and V. Vardanyan. Scalable Framework for Accurate Binary Code Comparison. In Proc. of the 2017 Ivannikov ISPRAS Open Conference, Moscow, 2017, pp. 34-38.
[23]. H. Aslanyan, S. Asryan, J. Hakobyan, V. Vardanyan, S. Sargsyan and S. Kurmangaleev. Multiplatform Static Analysis Framework for Programs Defects Detection. In CSIT Conference 2017, Yerevan, Armenia, 2017.
[24]. H.K. Aslanyan, S.F. Kurmangaleev, V.G. Vardanyan, M.S. Arutunian, S.S.Sargsyan. Platform-independent and scalable tool for binary code clone detection. Trudy ISP RAN/Proc. ISP RAS, vol. 1, issue 2, 2016. pp. 215-226 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-13.

Платформа межпроцедурного статического анализа бинарного кода

А.К. Асланян <hayk@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. В рамках данной статьи описывается разработанная платформа для статического анализа бинарного кода. Платформа разработана на основе

межпроцедурного, потоко-чувствительного и контекстно-чувствительного анализа программы. В качестве промежуточного представления используется машинно-независимый язык REIL. На этом представлении разработаны и реализованы основные анализы потока данных - анализ достигающих определений, построение DEF-USE и USE-DEF цепочек, трансформация для удаления мертвого кода, анализ значений, анализ помеченных данных, анализа памяти и т.д. Реализованный подход аннотации функций позволяет распространять данные между вызовами функций, тем самым сделав анализ чувствительным к контексту. Платформа предоставляет программный интерфейс для работы со всеми реализованным анализами, что позволяет добавлять новые анализы в качестве плагинов.

Ключевые слова: статический анализ, анализ бинарного кода, межпроцедурный анализ

DOI: 10.15514/ISPRAS-2018-30(5)-5

Для цитирования: Асланян А.К. Платформа межпроцедурного статического анализа бинарного кода. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 89-100 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-5

Список литературы

[1]. S. C. Misra and V. C. Bhavsar. Relationships between selected software measures and latent bug-density: Guidelines for improving quality. In Proc. of the International Conference on Computational Science and its Applications, ICCSA, Monreal, Canada, 2003.

[2]. V. P. Ivannikov, A. A. Belevantsev, A. E. Borodin, V. N. Ignatiev, D. M. Zhurikhin and A. I. Avetisyan. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, no. 5, 2014, pp. 265-275.

[3]. Coverity scan. Synopsys, https://scan.coverity.com/.

[4]. Klocwork static code analysis. RogueWave software, https://www.roguewave.com/products-services/klocwork/static-code-analysis.

[5]. Fortify Static Code Analyzer. Micro Focus, https://software.microfocus.com/ru-ru/products/static-code-analysis-sast/overview.

[6]. IBM AppScan. IBM, https://www.ibm.com/us-en/marketplace/ibm-appscan-source.

[7]. V. K. Koshelev, V. N. Ignatiev, A. I. Borzilov and A. A. Belevantsev. SharpChecker: Static analysis tool for C# programs. Programming and Computer Software, vol. 43, no. 4, 2017, pp. 268–276.

[8]. A. A. Belevantsev. Multilevel static analysis for improving program quality. Programming and Computer Software, 2017, pp. 321–336.

[9]. G. Balakrishnan and T. Reps. WYSINWYX: What You See Is Not What You eXecute. ACM Transactions on Programming Languages and Systems, vol. 32, no. 6, 2010, pp. 1-84.

[10]. H. J. Boehm. Threads cannot be implemented as a library. In Proc. of the 2005 ACM SIGPLAN conference on Programming Language Design and Implementation, 2005, pp. 261-268.

[11]. IDA Pro disassembler. Hex-Rays, https://www.hex-rays.com/products/ida.

[12]. Binnavi. Zynamics, https://www.zynamics.com/binnavi.html.

[13]. REIL - The Reverse Engineering Intermediate Language. Zynamics, https://www.zynamics.com/binnavi/manual/html/reil_language.htm.
[14]. R. E. Tarjan. Depth-first search and linear graph algorithms. In Proc. of the 12th Annual Symposium on Switching and Automata Theory, 1971, pp. 114 - 121
[15]. V. Aho, R. Sethi and J. D. Ullman. A formal approach to code optimization. In Proceedings of a Symposium on Compiler Optimization, 1970, pp. 86-100.
[16]. J. Kinder. Static analysis of x86 executables. Ph.D. Thesis, Technische Universitat Darmstadt, 2010.
[17]. S. Cheng, J. Yang, J. Wang, J. Wang and F. Jiang. LoongChecker: Practical Summary-Based Semi-simulation to Detect Vulnerability in Binary Code. *In* Proc. of the 10th International Conference on Trust, Security and Privacy in Computing and Communications, Changsha, 2011, pp. 150-159.
[18]. D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam and P. Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. In Proc. of the 4th International Conference on Information Systems Security, 2008, pp. 1-25.
[19]. D. Brumley , I. Jager , T. Avgerinos and E. J. Schwartz. BAP: A Binary Analysis Platform. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 463-469.
[20]. H. K. Aslanyan. Effective and Accurate Binary Clone Detection. Mathematical Problems of Computer Science, vol. 48, 2017, pp. 64-73.
[21]. G. S. Keropyan, V. G. Vardanyan, H. K. Aslanyan, S. F. Kurmangaleev and S. S. Gaissaryan. Multiplatform Use-After-Free and Double-Free Detection in Binaries. Mathematical Problems of Computer Science, vol. 48, 2017, pp. 50-56.
[22]. H. Aslanyan, A. Avetisyan, M. Arutunian, G. Keropyan, S. Kurmangaleev and V. Vardanyan. Scalable Framework for Accurate Binary Code Comparison. In Proc. of the 2017 Ivannikov ISPRAS Open Conference, Moscow, 2017, pp. 34-38.
[23]. H. Aslanyan, S. Asryan, J. Hakobyan, V. Vardanyan, S. Sargsyan and S. Kurmangaleev. Multiplatform Static Analysis Framework for Programs Defects Detection. In CSIT Conference 2017, Yerevan, Armenia, 2017.
[24]. А. Асланян, Ш. Курмангалеев, В. Варданян, М. Арутюнян и С. Саргсян, «Платформенно-независимый и масштабируемый инструмент поиска клонов бинарного кода,» Труды ИСП РАН, т. 28, № 5, pp. 215-226, 2016. DOI: 10.15514/ISPRAS-2016-28(5)-13.

Tracing ext3 file system operations in the QEMU emulator

V.M. Stepanov <vladislav.stepanov@ispras.ru>
P.M. Dovgalyuk <pavel.dovgaluk@ispras.ru>
D.N. Poletaev <poletaev@ispras.ru>
Yaroslav-the-Wise Novgorod State University,
11 Lasarevskaya Street, Velikiy Novgorod, Russia, 173000

Abstract. The paper proposes an approach to monitoring file operations through capturing virtual disk accesses in the emulator. This method allows obtaining information about file operations in the OS-agnostic manner but requires a separate implementation for each file system. An important problem for implementing this approach is the correct handling of changes in the file system. Operating systems that cache write requests can perform operations in any order. The authors have created a method for detecting read, write, create, delete and rename operations, and a module for QEMU, which monitors operations in the ext3 file system. The advantage of this method over others is that it does not interfere with the operation of the OS and does not depend on it. It is assumed that the QEMU module for file systems other than ext2/3 can be implemented using the methods described in this article.

Keywords: virtual machines; file systems; monitoring; QEMU; introspection

DOI: 10.15514/ISPRAS-2016-30(5)-6

For citation: Stepanov V.M., Dovgalyuk P.M., Poletaev D.N. Tracing ext3 file system operations in the QEMU emulator. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 101-108. DOI: 10.15514/ISPRAS-2018-30(5)-6

1. Introduction

The task of monitoring file operations is relevant when debugging the OS and its file system drivers, as well as researching the behavior of systems with an unknown internal organization, particularly performing the security audit of the information processed by such systems. The essence of the task is to display the actions and the names of the files with which the operations are performed.

Current solutions for file system monitoring are typically based on using the tools of the operating system and tracing system calls. These solutions differ depending on the operating system, and some exotic OSes might not have the appropriate tools for this task.

The approach described in this article does not require any knowledge about the operating system used. The information about file system operations is obtained

through capturing the disk requests of the virtual machine. The implementation that has been created is based on the QEMU emulator [1]. By modifying the source code of the project, functionality has been added to monitor and log the file operations of the system.

A data read or write query contains the disk sector number and the number of bytes to read or write. The QEMU module identifies the file names based on the sector in the query, the virtual disk information and the knowledge about the structure of the file system. Every file system type has its own distinctive internal organization different from others and thus requires its own implementation of the module. As an example, a monitoring tool for the ext3 file system [2] has been implemented, which is one of the file systems used in Linux-based operating systems.

As a result of this project, a module was created to monitor file operations of any guest OS, but only if it uses an ext3 file system. It is expected that the ideas used in this implementation can be applied to other file systems as well.

2. Overview of existing solutions

First, the authors would like to review several tools for file system monitoring.

Inotify is a Linux kernel subsystem intended for monitoring file system events [3]. This mechanism can be used to monitor such file operations as reading, creating, deleting, changing files, etc., by subscribing to events. The application creates an inotify object and informs the kernel about the files needed. The kernel responds by sending notifications, which can be received by the application by reading the file. Users can monitor the activity of the file system by using command-line utilities from the inotify-tools library.

Other operating systems have similar mechanisms. For example, Windows uses FileSystemWatcher [4]. FreeBSD and Mac OS X allow monitoring changes using kqueue [5].

QEMU-Based Framework for Non-intrusive Virtual Machine Instrumentation and Introspection [6] is a system that uses a binary application interface to analyze the state of the virtual machine. The system includes a file monitoring plugin, which receives information about file operations by capturing the corresponding system functions. Since these system calls are different for different operating systems, a specific plugin is created for each operating system. Currently, file monitoring is implemented for Windows and Linux. Unlike the preceding mechanisms, this tool allows monitoring the file operations of the virtual machine without interfering with the guest operating system processes.

The proposed approach, like the plugin described above, does not require modifying the operating system. The difference is that the implementation of this approach does not have any dependencies on the operating system. Instead, it depends on the file system.

One of the possible use cases of this project implies monitoring file operations in exotic file systems where information about system calls is not present or which do

not have system calls in their traditional sense because the whole system operates in a privileged mode. The information about what files are being used in such an operating system can be useful for analysis.

3. Ext3 file system

The authors will now briefly describe the structure of the ext3 file system, which the tool is intended for. The space of the file system is divided into fixed-size blocks. For the purposes of optimization, the blocks are combined into groups. Each group has a description block, bit masks, and an inode table (fig. 1).

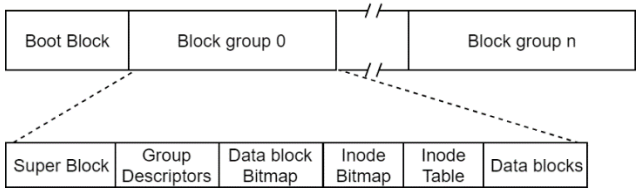

Fig. 1. Ext3 file system structure

An inode is a structure which contains the addresses of all blocks of the file, as well as its attributes, such as the file type and access permissions. The name of the file is contained in a separate structure – the parent directory. The directory contains a table, in which each of the entries represents a child file and includes the name and number of the respective inode.

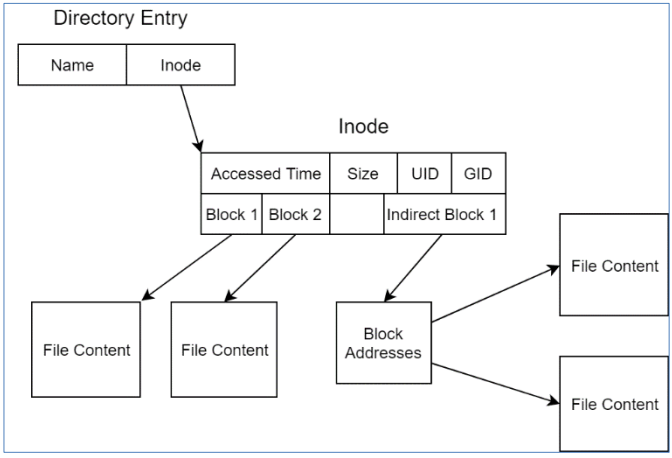

Fig. 2. General information structure of the ext3 file system: directories, inodes, and data blocks

The physical location of the file is represented as data block numbers in the inode. If any inode does not have sufficient space to store all the addresses of the data blocks, references to additional address blocks are used. Address block entries can refer to

other address blocks, and so on. In this case, such address blocks are called double and triple indirect blocks. The interconnections of the described file structures are depicted in fig. 2.

When opening a file, the OS uses the path name to access the data. The directory system converts the ASCII name into information needed to find the data. However, for the purposes of monitoring file operations, the authors are interested in the reverse process. A method is needed that will make it possible, by using a specific sector number, to obtain the name of the corresponding file.

4. Possible ways of finding the file name based on its sector number

The task to find the full file name based on one of its sector numbers has several possible solutions.

The first way implies that every disk operation should be accompanied by iterating through all the files and the addresses of their data blocks until the file with the particular sector is found. The time to complete such iteration is, in the worst case, linearly dependent on the space used in the partition.

Implementing this solution showed that for the 6.0 GB disk with 4.0 GB used, in an ext3 file system, one such disk query may take up to 12 seconds. This time measurement was performed on a computer configuration using an AMD FX-8370E processor and 16 GB of RAM. Thus, it may be concluded that to achieve high performance, the number of full file system iterations should be minimized.

Another way to do this implies creating special data structures with the aim to achieve higher searching speed. A directory tree with file names and an associative array with fast search functionality allow completing this task in logarithmic time. It is only required for the keys of the associated array to be the block numbers, and for the values to be the names of the respective files in the directory tree.

Creating these fast search structures is performed on capturing the first query to the partition. The question of how these structures should change while capturing new operations in the file system has several answers.

The first option is: the structures can remain unchanged. In this case, file accesses performed before the operating system has been loaded can also be traced. However, changing the size and location of these files makes the output data about file operations irrelevant or incomplete.

The second option is: the structures can be rebuilt from scratch with some determined periodicity. In this case, operations with new and changed files can be traced. However, some situations are possible when an operation with a recently created or enlarged file takes place, but the fast search structures have not yet been updated. In this case, such an operation can be left untraced.

The third option is: the structures can be rebuilt after writing to index descriptors, address blocks, and directories. This allows the fast search structures to always correctly reflect the current file system state. However, this method leads to a

significant increase in time needed to process all the operations that change the file system state, which can negatively affect the performance of the guest OS.

The fourth option implies that when operations that change the file system occur, the existing fast search structures also change. With this method, fast processing of any disk accesses and monitoring of the current information about file operations can be achieved. The drawback of this solution is its implementation complexity.

Operations that change the file system include creation, deletion, expansion, truncation, renaming and moving of files. The mechanism to recognize these operations is based on detecting the structure to which data is written and then comparing old data with new. For example, if an operation results in adding a new entry to a directory, it indicates that a new file was created, or that an old file has been moved to this directory. For each type of operations, a specific processing mechanism exists, which performs required changes to the directory tree and the associated array.

5. The problem of unspecified order of queries to the disk

There is a problem which results in some file operations not being detected using the described methods. The problem is that most OSes do not perform writing to the disk immediately but delay it for a period of time. At the same time, the order in which the pending writing operations will eventually be performed can vary. In some cases, the information is registered in structures even though no information about their existence has been written to the disk yet.

The time diagram in fig. 3 demonstrates one of the possible sequences of operations when writing data to a new file.

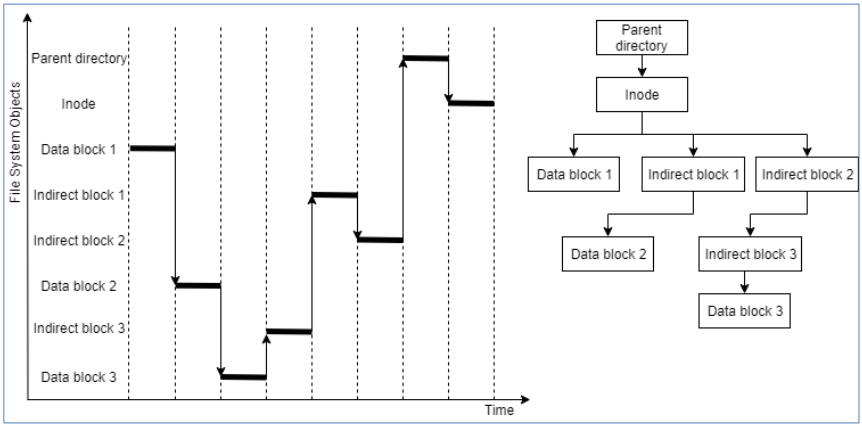

Fig. 3. Time diagram of writing data in case of new file creation

The solution to this problem is to store the information about unidentified operations for some time. When adding addresses to an inode and indirect blocks, previously unidentified operations are checked. If the address of the new block matches the destination of any of the earlier operations, this operation is processed.

A separate case occurs when creating files. In this situation, writing to the parent directory and the inode can be performed in any order. If, at the moment of writing to the directory, writing to the inode has already taken place, then this file operation is processed. Otherwise, the processing will be postponed until writing to the inode.

One more case is file moving. Sometimes the file information is added to the new directory first and is removed from the old one only after that. In this case, the file is moved in the directory tree while the deletion is ignored.

6. Testing

The implemented module creates its additional structures once during its operation, and then changes them in accordance with the new file system state. To verify that the module operates as expected, a series of tests were conducted using different guest OS images. The tests involved opening applications, navigating from folder to folder, as well as reading, creating, changing, deleting, renaming and moving files. Various situations were checked when the order of operations is ambiguous.

The following is an example of the module operation. In the guest OS (in this case, Debian), the command “dd if=test of=test1” is performed. This command copies the data in file “test”, 1 Kb in size, into file “test1”. The entries added to the log file are presented in fig. 4.

```
read   6926160      16384   /bin/dd
read   6926192      32768   /bin/dd
read   6926256      12288   /bin/dd
read   10263952     4096    /home/debian/test
...
create /home/debian/test1
write  2640360      4096    /home/debian
write  10490400     4096    /home/debian/test1
```

Fig. 4. Fragment of a log file generated by the module

Log entries contain information about the operation type, the sector number, the number of bytes affected and the file name. In this case, first, the executable file “dd” from the directory “bin” is read, and then the file “test” is read. Approximately 20 seconds after the input of the command, pending operations of writing to the disk are performed, and the log file is updated with new events. Then, the file “test1” is created, which is populated with data from “test”.

The testing was performed using Linux, FreeBSD, Windows 10 and KolibriOS operating systems [7]. The tests show that the module successfully registers file operations and processes the file system changes. At the same time, no lags due to the module monitoring were observed.

KolibriOS was chosen for testing as an example of an exotic operating system. This is a miniature OS, with its core and most of its programs written in assembly language. While testing, it was found that writing operations in this OS are not cached but performed immediately. Consequently, the problem of the order of operations being ambiguous is irrelevant for KolibriOS.

Thus, it was verified that the module functions correctly with each of the tested OSes.

7. Conclusion

In this paper, an approach to file operations monitoring has been described. This approach allows analyzing operations with operating system files and applications in a virtual machine. The implemented module works with the ext3 file system. It is intended for capturing virtual disk accesses in the guest system and writing the operation type and file name into a log. In contrast to internal file system monitoring tools, such as inotify, the created QEMU module can monitor file operations without interfering with the operation of the guest OS. In addition, the module does not depend on system calls, which allows it to work with any OS. While implementing the module, it has also been made possible to achieve a high speed of file operations processing. To do this, QEMU creates and maintains special structures: binary search trees and directory trees. The solutions described in the article can be applied to develop monitoring instruments in other file systems, including FAT32, NTFS, ext4.

References

[1]. Bellard, F. QEMU, a fast and portable dynamic translator. In Proceedings of the USENIX Annual Technical Conference, 2005, pp. 41–46.
[2]. Brian Carrier, File System Forensic Analysis. Addison-Wesley Professional, 2005.
[3]. Koen Vervloesem. Inotify: Watch your filesystem. Linux format, № LXF140, 2011.
[4]. FileSystemWatcher. https://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher(v=vs.110).aspx
[5]. Jonathan Lemon. Kqueue - A Generic and Scalable Event Notification Facility, Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, 2001, p.141-153
[6]. P. Dovgalyuk, N. Fursova, I. Vasiliev, V. Makarov. 2017. QEMU-based framework for non-intrusive virtual machine instrumentation and introspection. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), pp. 944-948. https://dx.doi.org/10.1145/3106237.3122817
[7]. Artem Jerdev. Kolibri-A: a lightweight 32-bit OS for AMD platforms, Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), pp. 20-22.

Отслеживание операций с файловой системой ext3 в эмуляторе QMU

В.М. Степанов <vladislav.stepanov@ispras.ru>
П.М. Довгалюк <pavel.dovgaluk@ispras.ru>
Д. Н. Полетаев <poletaev@ispras.ru>
Новгородский государственный университет имени Ярослава Мудрого, 173000, Россия, г. Великий Новгород, ул. Лазаревская, дом 11

Аннотация. В работе рассматривается подход к отслеживанию файловых операций с помощью перехвата запросов к виртуальному диску в эмуляторе. Такой способ

позволяет получать информацию о файловых операциях независимо от гостевой ОС, однако требует отдельной реализации для каждой файловой системы. Важной проблемой для реализации данного подхода является корректная обработка изменений в файловой системе. С операционными системами, которые имеют свойство кешировать операции записи, возникают осложнения, так как операции записи могут выполняться в произвольном порядке. Для примера подхода был создан модуль эмулятора QEMU, отслеживающий операции с файловой системой ext3. Преимущество данного инструмента перед другими состоит в отсутствии вмешательства в работу ОС, а также отсутствии зависимости от ОС. Благодаря этому возможно использование на таких экзотических ОС, с которыми не работают другие инструменты мониторинга файловых операций. Предполагается, что модуль QEMU для файловых систем, отличных от ext2/3, может быть реализован с использованием методов, подобных описанным в статье.

Ключевые слова: виртуальные машины; файловые системы; мониторинг; QEMU; интроспекция

DOI: 10.15514/ISPRAS-2018-30(5)-6

Для цитирования: Степанов В.М., Довгалюк П.М., Полетаев Д.Н.. Отслеживание операций с файловой системой ext3 в эмуляторе QMU. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 101-108 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-6

Список литературы

[1]. Bellard, F. QEMU, a fast and portable dynamic translator. In Proceedings of the USENIX Annual Technical Conference, 2005, pp. 41–46.
[2]. Brian Carrier, File System Forensic Analysis. Addison-Wesley Professional, 2005.
[3]. Koen Vervloesem. Inotify: Watch your filesystem. Linux format, № LXF140, 2011.
[4]. FileSystemWatcher. https://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher(v=vs.110).aspx
[5]. Jonathan Lemon. Kqueue - A Generic and Scalable Event Notification Facility, Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, 2001, p.141-153
[6]. P. Dovgalyuk, N. Fursova, I. Vasiliev, V. Makarov. 2017. QEMU-based framework for non-intrusive virtual machine instrumentation and introspection. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), pp. 944-948. https://dx.doi.org/10.1145/3106237.3122817
[7]. Artem Jerdev. Kolibri-A: a lightweight 32-bit OS for AMD platforms, Postgraduate Conference for Computing: Applications and Theory (PCCAT 2011), pp. 20-22.110

Reading the contents of deleted and modified files in the virtualization based black-box binary analysis system Drakvuf

S.G. Kovalev <skovalev@ptsecurity.com>
Positive Technologies
8 Preobrazhenskaya Square, Moscow, 107061, Russia

Abstract. The article discusses ways to get the content of files, which are modified during the processing in the well-known open source dynamic analysis environment Drakvuf. Drakvuf initially implemented file saving functionality based on the use of undocumented mechanisms for working with the system cache. The author of this article proposes a new approach to obtaining the content of files on Microsoft Windows family systems using Drakvuf. The proposed approach is based solely on the use of the public interface of the kernel by the hypervisor and provides portability between different versions of the operating system. In the conclusion of the article, the advantages and disadvantages of both approaches are presented, and directions for further work are proposed.

Keywords: malware; dynamic analysis; injection; Drakvuf; Virtual Machine Introspection.

DOI: 10.15514/ISPRAS-2016-30(5)-7

For citation: Kovalev S.G. Reading the contents of deleted and modified files in the virtualization based black-box binary analysis system Drakvuf. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 109-122. DOI: 10.15514/ISPRAS-2018-30(5)-7

1. Introduction

In recent years, a steady increase in the number of malicious programs has been registered [1]. A direct consequence was the impossibility of manual analysis of this thread, which led to the emergence of the need for scalable and automated tools for collection and analysis of malware. Such tools include honeypots [2, 3] and sandboxes [4, 5]. At the same time, it is worth noting that malware uses various techniques to detect analysis tools [6], which imposes large restrictions on such tools.

The use of virtual machine monitors provides several advantages for creating such tools: isolation of a program of interest, the ability to quickly and easily restore a compromised system, as well as scalability. Dynamic analysis requires the completeness and accuracy of collected data. The use of virtual environments also allows meeting these requirements, providing an analysis environment with information about code execution, disk and memory usage in real time.

There are at least two approaches to the use of virtualized environments for dynamic analysis. In the first case, the virtual machine is used as a replacement for the physical one, which allows for scalability and a greater speed of restoring the analysis environment; wherein, the kernel driver or injection of DLL into the address space of a process of interest is used for the analysis. In the second case, the virtual machine monitor is expanded with new possibilities of studying the virtual machine state and does not require the installation of additional software in the system of interest. This approach is called "virtual machine introspection" [7] or VMI.

One of the important components of dynamic analysis is reading the contents of deleted and modified files. This is due to the fact that some malicious programs download the payload over the network and save it in a temporary file. This class of malware was named Trojan Downloader [8]. Reading the contents of a newly created file is necessary for further analysis. Another class of malware, called Ransomware Trojans [9], encrypts many files on the disk. The presence of information about a large number of newly created files with similar names or about modified contents is a necessary condition for detecting malicious behavior.

Further, this article discusses the dynamic analysis environment Drakvuf [10, 11], and two ways to read the contents of files. The first method was present initially and was built on the knowledge of internal structures of the operating system kernel. The second method was added by the author of this article and is based on the injection of system functions. Thus, this approach relies on the stable public API of the operating system kernel and in some cases allows reading the full contents of files.

2. Overview of the dynamic analysis environment Drakvuf

Drakvuf is a virtualization based agentless black-box binary analysis system based on «virtual machine introspection».

To build this environment, the following solutions were used:

- Xen virtual machine monitor [12];
- LibVMI library [13], which allows access to the low-level state of a virtual machine;
- Rekall framework for studying the virtual memory of an operating system of interest [14].

Further, each of the components and the way to use them together with Drakvuf are discussed.

2.1. Xen

Xen is a bare-metal (i.e. independent of the operating system) hypervisor that supports hardware virtualization technology. Xen allows running multiple virtual machines, the so-called DomU. In this case, one of them is considered to be controlling, the so-called Dom0.

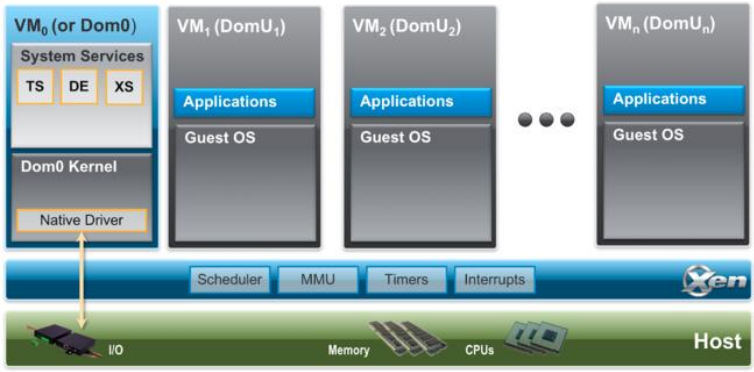

Fig. 1. Xen architecture

Xen provides resource allocation for virtual machines, scheduling of virtual kernels, and interrupt control. Dom0 is used to interact with the user, providing the system with external device drivers (NIC, SATA, etc.). Dom0 typically runs the QEMU process [15] associated with each virtual machine. QEMU provides emulation of a virtual machine target platform (system logic set, BIOS or UEFI, external devices). QEMU execution is supported in dedicated domains, the so-called subdomains, which increases safety and performance.

Starting from Xen 4.5, the API for VMI is added to the hypervisor. Subsequently, this interface is constantly being improved.

2.2. LibVMI

LibVMI is a library that provides access to the state of a virtual machine. It provides the following capabilities (the list is not complete):

- reading and changing the contents of the virtual memory of a VM of interest;
- setting permissions to the physical memory of a VM;
- reading and changing the values of VM processor registers;
- stopping and resuming VM operation;
- installing handlers for certain hardware events in a VM:
    - changing values of control registers (CR0, CR3, CR4);
    - access violations to the physical memory of a VM;
    - single-step debugging of the VM execution;
    - debugging interrupt (INT 3).
- LibVMI uses Xen VMI API for hidden analysis and change of the VM state.

2.3. Rekall

Rekall is a virtual memory analysis framework. In the context of Drakvuf, the feature of building an operating system profile based on debugging symbols of a specific operating system is promising.

For example, for operating systems of the Microsoft Windows family, symbols of the kernel and main modules are provided in the PDB format. Rekall allows converting a PDB file to the JSON format. Such JSON file is called a profile [16] and contains the following information:

- a brief description of the kernel for which the profile has been compiled (family, version, build number);
- a list of constants and their offsets in the kernel;
- description of structures (names of members and their offsets within structures).

The presence of such information allows overcoming the semantic gap between the analysis environment and a system of interest.

2.4. Drakvuf

Drakvuf combines the ability to analyze and change the state of the VM provided by LibVMI and the debugging information provided by Rekall with the knowledge of the internal structure of an operating system of interest. This allows achieving the following features:

- detection of the current process and thread at an arbitrary point in time;
- detection of the virtual address of a symbol (of constant or function) by name;
- setting a virtual address trap;
- getting the file name according to the file handle.
- In addition, Drakvuf provides a plugin architecture and an initial set of plugins. Plugins include the following ones:
- syscalls – allows tracking entry points to system call handlers;
- filedelete – allows reading the contents of deleted and modified files.
- In the presented work, the filedelete plugin has been significantly improved, as will be discussed in section 5.
- It is important to note that all useful activities are performed during the processing of exit from a VM (the so-called “VM exit”). Thus, the Drakvuf operation scheme is as follows:
- at the very beginning of Drakvuf operation, the VM is stopped;
- traps and event handlers are configured (in plugins);
- the main loop is started:
    - VM operation is resumed, and Drakvuf begins to wait for notification of an event;
    - one of the expected events occurs in the VM;

- Xen stops the VM and transfers control to Dom0, where Drakvuf is usually running;
- Drakvuf (LibVMI) bypasses the list of handlers for events of this type, transferring the control to each of them by rotation.

2.5. Using the Drakvuf trap mechanism to determine deleted and changed files

To determine deleted and changed files, traps on the following system functions are installed from ntoskrnl.exe:

- NtSetInformationFile – is used to delete a file when closing the last file handle;
- NtWriteFile – records data to a file;
- NtClose – closes the file handle.

The NtWriteFile handler adds the following data to the list: PID of the process, file handle, and file name. The NtClose handler for modified files removes an entry from the list and proceeds to reading the contents of a file. The NtSetInformationFile handler proceeds to read the contents of any deleted file.

3. Reading the contents of files by analyzing the cache manager

For a detailed presentation of the material see [17]. The following is a general description of the approach which is necessary and sufficient for comparison.

In the beginning, the _FILE_OBJECT structure location is determined according to the file handle. Next, using the value of a member of the SectionObjectPointer structure, the location of the _SECTION_OBJECT_POINTERS structure is determined, the DataSectionObject member of which points to the _CONTROL_AREA structure. At the end of this structure, there is the first member of a linked list, consisting of _SUBSECTION structures. Each such structure defines a sequential memory chunk mapped to a file. Having read the contents of all such chunks, one can compile a file (or at least part of it, see below).

In the _SUBSECTION structure, the following members are significant:

- SubsectionBase – the first member of the array of _MMPTE entries, each of which defines the physical address of a page (in terms of VM) and some flags;
- PtesInSubsection – the number of array members;
- StartingSector – the offset of the first page of this section in the file, expressed in chunks of 512 bytes.

Each _MMPTE entry is a 4 KB virtual memory page descriptor (the so-called PTE, or “page table entry”). Collectively, PTEs describe a continuous virtual memory block that represents a portion of the file starting from the StartingSector*512 offset. However, some pages can be paged out from the RAM of the VM. This is indicated by the zero value of the Present flag in PTE.

Thus, in order to read the cache manager contents, Drakvuf just needs to bypass the list of PTEs for each section and to read the contents of each page for which the Present flag is set.

3.1. Limitations of this approach

Although this approach provides reading of the cache manager content in a way that is fast and invisible for the VM, it has several limitations:

- large files can be accessed in parts; in this case, the cache manager may contain one or more fragments of the file, while the rest of the file even will not be downloaded;
- the current implementation does not take into account the fact that some pages with cleared Present flag may still contain data not downloaded to disk;
- memory for cache manager structures is allocated from the system working set and can be paged out to disk;
- the current implementation does not support working with memory-mapped files [18].

The above limitations led to the beginning of work on injection of system functions to read the contents of files.

4. Injection of system calls

Initially, the linbinjector library was added to Drakvuf, which provided an injection of the CreateProcess system function. This allowed for the direct launch of an application of interest in the VM, requiring only the presence of a file on the VM disk. This approach (the so-called agent-free approach) provides greater secrecy compared to the classical solution in which the remote control process was launched in the VM. Since the injection of functions is an integral part of the proposed solution, here is a general description of the approach.

The function injection implies a change in the state of the current instruction stack and register (IP on x86 architecture), which emulates the sequence of operations used by the compiler when calling a function (the word “call” can be further used instead of the word “injection”).

Since operating systems of the Microsoft Windows family are considered, the rules for calling functions in the kernel are well documented [19]. For example, let us consider the injection of the ZwQueryVolumeInformationFile function call on a 64-bit system.

This function takes five arguments: object handle (integer), pointer to the IO_STATUS_BLOCK structure, pointer to the FILE_FS_DEVICE_INFORMATION out structure (for the example), size of out structure, structure type (integer, for FILE_FS_DEVICE_INFORMATION is 4). In accordance with the accepted ABI, the first four arguments are transferred in RCX, RDX, R8, R9 registers, and the last argument is transferred on the stack.

However, there are some limitations that shall be considered:

- before calling a function on the stack, space for four arguments is reserved (the so-called “home space”);
- when transferring a pointer to a structure, the address of the structure beginning must be aligned with a value equal to the greatest alignment of any member of the structure;
- before calling a function, the stack shall be aligned by a multiple of 16 B.

The last two requirements were not initially taken into account, which led to a time-consuming debugging of various fatal kernel errors (the so-called BSOD).

After all arguments of the function are prepared, the return address is set on the stack. As a rule, it coincides with the trap address, which allows continuing execution of the VM. In this case, the trap is not deleted, which is necessary for processing the exit from ZwQueryVolumeInformationFile.

Lastly, the ZwQueryVolumeInformationFile address is entered to the RIP register and the VM operation is resumed.

Since it is possible to setup new trap, after the ZwQueryVolumeInformationFile function completes, the trap handler receives control again, which allows processing returned data, restoring registers and the stack, and continuing operation of the VM.

Further development of this approach led to the idea of the possibility of sequential execution of several injections, which allowed reading the contents of files without reference to the cache manager structures.

5. New approach to reading the contents of files by injection of system calls

The proposed approach was a direct consequence of the desire to achieve guaranteed reading of the contents of arbitrarily large files, not limited to what is contained in the cache manager. The kernel already provides the ZwReadFile system function. However, one cannot simply call ZwReadFile on the handle of an arbitrary object:

- the handle can be linked with a logical disk volume, I/O device, etc.;
- to read files, one needs to prepare a memory buffer of sufficient size;
- for files that do not fit into the buffer, several read operation calls are required;
- reading of asynchronous files can lead to unexpected errors.

In the course of the work, the author discovered at least two more circumstances that were not initially taken into account:

The stack size in the kernel mode is limited (16 KB for 32-bit systems and 24 KB for 64-bit systems), so it is impossible to reliably allocate a sufficiently large memory buffer on the stack;

in a multithreading OS, a process or a thread may switch while the contents of the file are being read;

in LibVMI, all trap handlers registered to a virtual address are sequentially called, so it is necessary to distinguish the beginning of the chain from its middle.

Here is the brief description of solutions for each of these limitations.

5.1. Solving the problem of determining the type of the handle

The ZwQueryVolumeInformationFile function called with the FileFsDeviceInformation parameter returns the FILE_FS_DEVICE_INFORMATION structure. The first member of this structure DeviceType takes one of the values [20]. During the research, it was revealed that regular files are of FILE_DEVICE_DISK type (i.e., 0x7).

5.2. Solving the problem of buffer preparation

In the beginning of work, the author did not take into account the fact, that the kernel stack size is not only limited but also rather small (16 kB for 32-bit systems and 24 kB for 64-bit systems). Thus, in the first version, the 4kB buffer was allocated directly on the stack. However, the author has soon noticed that in some cases OS has a fatal error when reading a file. It was suggested that the reason is a kernel stack overflow.

In order to eliminate such an error, it was decided to allocate the buffer in a non-paged memory area (so-called «NonPaged Pool»). This provides an additional advantage of the possibility to allocate more memory (for example, 64 KB).

For further optimization, the allocation of new memory buffer on request was added. All allocated memory buffers are put in the list. Initially, the list is empty. Each new thread first accesses the list. If there is a free memory buffer in the list, it is marked as busy and used for reading operation. If there are no free buffers in the list, the ExAllocatePoolWithTag function is called (injected) first.

In practice, it turned out that a single memory buffer is sufficient for a VM with two kernels.

5.3. Solving the problem of reading large files

In practice, there are often large files that do not fit in one memory buffer. Therefore, it becomes necessary to perform the file read operation in a loop. However, the file size is not known in advance. It would be possible to use one of the system functions to read the file size, but this would extend the call chain and reduce system performance. In addition, there is a need to move the carriage in the file. Fortunately, the ZwReadFile function already has all the necessary properties to solve this problem.

One of the ZwReadFile arguments is a pointer to the IO_STATUS_BLOCK structure. Upon the completion of the read operation, this structure contains two members: the operation completion code and the number of bytes read.

The second useful argument in the context of this task is the ZwReadFile argument, which is a pointer to the LARGE_INTEGER ByteOffset structure. This argument

allows setting the offset in the file from which ZwReadFile will start reading the contents.

Using the second member IO_STATUS_BLOCK and ByteOffset allows creating a simple read algorithm for a large file: as long as the read operation returns STATUS_SUCCESS and the number of bytes read is equal to the size of the transferred memory buffer, continuing the read operation, increasing the offset by the memory buffer size. Wherein, at the beginning it is necessary to explicitly specify a zero offset, because in practice, at the time of calling NtClose, the carriage was shifted to the end of the file. This resulted in a read error STATUS_END_OF_FILE.

5.4. Solving the problem of asynchronous files

At the beginning of the research, it was noticed that ZwReadFile often returns the STATUS_PENDING error code. This means that an attempt to read a file opened for asynchronous access is being made [18]. The first solution was to add a call to the WaitForSingleObject function. This call is different from others. There was the need to keep the stack from the previous call ZwReadFile and the lack of its own handler. The only thing that the trap handler did on WaitForSingleObject was transferring control to the ZwReadFile handler, which again checked the error code and read the memory buffer.

However, it soon became clear that the operation of the system became unstable. Often there were fatal kernel errors associated with breaking the stack. Debugging of the kernel showed that in almost all cases the stack pointer was more than 1 MB from the base of the nuclear stack (so-called “stack underflow”). A further study of the stack showed that the violation of the stack began with calling ZwReadFile. It was not possible to establish the exact cause of the error, but there was a clear dependence of the error reproducibility on the type of files read. Errors were reproduced when accessing asynchronous files.

Thus, it was decided not to attempt to read such files. Finding out whether the file was open for asynchronous access turned out to be trivial. The _FILE_OBJECT structure contains the Flags member. If the FO_SYNCHRONOUS_IO flag is set, the file has been opened for synchronous access. So it is possible to read its contents.

This simple revision led to an increase in the reliability of the entire system. However, the issue of reading the contents of files opened for asynchronous access remained open. The answer to this question is partly given below.

5.5. Solving the problem of processing several traps at one virtual address

The need to process the returned values of called functions results in at least two handlers at one virtual address: a constant handler at NtClose and a temporary handler for the function being called. The situation is aggravated by the fact that the

handler of each stream can be installed to the same address. Thus, it is necessary not only to distinguish the beginning of a call chain but also to distinguish between processes and threads. Moreover, it turned out that all handlers of traps to a given virtual address are traversed in LibVMI. Therefore, upon completion of the read operation, one cannot simply delete a trap. This will lead to looping attempts to read the file.

To solve this problem, a map was added, which maps a pair of process and thread values to a marker for completion of a read operation. When a trap on NtClose is triggered and a decision is made to read the contents of a file, a new process thread pair is added to the map with an empty marker. When a file read operation is completed in any form, a marker in the map for the current process thread is filled, and the trap is deleted. Since in LibVMI new traps are added to the top of the list, for the current process thread a trap on NtClose is executed the last. It checks the marker and, if it is full, the entry is deleted from the map, and the handler ends.

At the same time, the handler of each called function checks the compliance of the current process-thread with the stored value, which eliminates the accidental triggering of the handler.

5.6. Solution algorithm

By putting together all of the above, the following file reading algorithm is obtained:

- Step 1. Check that the file is open for synchronous access, otherwise shut down.
- Step 2. Check that no read operations are performed for the current process-thread and add a marker to the map, otherwise remove the marker from the map.
- Step 3. Call ZwQueryVolumeInformationFile and check that the regular file is processed, otherwise fill in the marker and complete the work.
- Step 4. Allocate a memory buffer if there is a free one, otherwise call ExAllocatePoolWithTag.
- Step 5. In the loop, call ZwReadFile as long as the error code is STATUS_SUCCESS and the number of bytes read is equal to the size of the memory buffer.
- Step 6. Fill in the marker for the current process thread.

If one of the steps fails by mistake, the attempt to read the file is considered failed, and an attempt to read parts of the file from the cache manager is made. This partly solves the problem with asynchronous files.

Thus, the proposed approach significantly expanded the existing one, allowing reading the contents of files reliably, using the documented system functions.

6. Conclusion

The paper presents a new approach for reading the contents of deleted and modified files during automated dynamic analysis of applications on Microsoft Windows operating systems. This approach has a distinctive feature of using the mechanism for injecting system functions of the operating system running in a virtual machine from the side of the hypervisor. This technique avoids the presence of agent applications or drivers in the virtual machine and increases secrecy, which is extremely important in studying the malware. It uses documented system functions, which allows achieving transferability between different versions of operating systems of this family.

The problem of reading the contents of files opened for asynchronous access is not fully solved, which sets the direction for further activities.

In addition, this paper provides an overview of the dynamic analysis environment Drakvuf, its constituent parts and some principles of work. It considers the initial approach to reading the contents of files based on reading internal structures of the cache manager, and its limitations.

References

[1]. The Independent IT-Security Institute. Malware. Available at: https://www.av-test.org/en/statistics/malware/, accessed 17.11.2018.

[2]. Asrigo K., Litty L., Lie D. Using VMM-Based Sensors to Monitor Honeypots. Department of Electrical and Computer Engineering University of Toronto, 2006. Available at: https://security.csl.toronto.edu/papers/asrigo-vee2006.pdf, accessed 17.11.2018.

[3]. Rangian M.K., Attri U. Design and Implementation of Malware Collection System Based on Client Honeypot. International Journal of Scientific & Engineering Research, vol. 4, issue 3, 2013, pp. 775-780.

[4]. Cuckoo Sandbox. Available at: https://cuckoosandbox.org/, accessed 17.11.2018.

[5]. Willems C., Holz T., Freiling F. Toward Automated Dynamic Malware Analysis Using CWSandbox. IEEE Security & Privacy, vol. 5, issue 2, 2007, pp. 32-39.

[6]. Malware Anti-Analysis Techniques and Ways to Bypass Them. Available at: https://resources.infosecinstitute.com/malware-anti-analysis-techniques-ways-bypass/, accessed 02.05.2017.

[7]. Garfinkel T., Rosenblum M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. Computer Science Department, Stanford University, 2003. Available at: https://suif.stanford.edu/papers/vmi-ndss03.pdf, accessed 17.11.2018.

[8]. Kaspersky Lab. Malware Classification (in Russian). Available at: https://www.kaspersky.ru/blog/klassifikaciya-vredonosnyx-programm/2200/, accessed 17.11.2018.

[9]. Symantec Corporation. What Is Ransomware? Available at: https://us.norton.com/internetsecurity-malware-ransomware.html, accessed 17.11.2018.

[10]. Drakvuf. Available at: https://drakvuf.com/, accessed 17.11.2018.

[11]. Lengyel T.K. Malware Collection and Analysis via Hardware Virtualization. University of Connecticut, 2015. Available at: https://tklengyel.com/thesis.pdf, accessed 17.11.2018.

[12]. Xen Project. Available at: https://xenproject.org/, accessed 17.11.2018.
[13]. LibVMI. Available at: http://libvmi.com/, accessed 17.11.2018.
[14]. Rekall Forensics. Available at: http://www.rekall-forensic.com/, accessed 17.11.2018.
[15]. QEMU. Available at: https://www.qemu.org/, accessed 17.11.2018.
[16]. Rekall Profiles. Available at: http://blog.rekall-forensic.com/2014/02/rekall-profiles.html, accessed 17.11.2018.
[17]. Russinovich M., Solomon D., Ionescu A. Microsoft Windows Internal Design. The Main OS Subsystems, 6th ed. (in Russian). Saint Petersburg, Piter, 2014, 672 p.
[18]. Richter J., Nazar C. Windows via C/C++. Visual C++ Programming (in Russian). Saint Petersburg, Piter, 2009, 896 p.
[19]. Building C/C++ Programs. Available at: https://docs.microsoft.com/en-us/cpp/build/building-c-cpp-programs?view=vs-2017, accessed 17.11.2018.
[20]. Specifying Device Types. Available at: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/specifying-device-types, accessed 17.11.2018.

Получение содержимого удаляемых и изменяемых файлов в среде динамического анализа исполняемых файлов Drakvuf

С.Г. Ковалёв <skovalev@ptsecurity.com>
Positive Technologies
107061, Москва, Преображенская пл., д. 8

Аннотация. В статье рассматриваются способы получения содержимого файлов, изменяемых в процессе работы известной среды динамического анализа с открытым исходным кодом Drakvuf. В Drakvuf изначально реализована функциональность сохранения файлов, основанная на использовании недокументированных механизмов работы с системным кэшем. Автором данной статьи предложен новый подход получения содержимого файлов в системах семейства Microsoft Windows с помощью Drakvuf. Предложенный подход основан исключительно на использовании публичного интерфейса ядра со стороны гипервизора и обеспечивает переносимость между различными версиями операционной системы. В завершение статьи приведены достоинства и недостатки обоих подходов, предложены направления дальнейших работ.

Keywords: вредоносная программа; динамический анализ; инъекция; Drakvuf; Virtual Machine Introspection.

DOI: 10.15514/ISPRAS-2018-30(5)-7

Для цитирования: Ковалёв С.Г. Получение содержимого удаляемых и изменяемых файлов в среде динамического анализа исполняемых файлов Drakvuf. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 109-122 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-7

Список литературы

[1]. Malware. The Independent IT-Security Institute. Доступно по ссылке: https://www.av-test.org/en/statistics/malware/.

[2]. Kurniadi Asrigo, Lionel Litty, David Lie. Using VMM-Based Sensors to Monitor Honeypots. Department of Electrical and Computer Engineering University of Toronto, 2006. Доступно по ссылке: https://security.csl.toronto.edu/papers/asrigo-vee2006.pdf, дата обращения 17.11.2018.
[3]. Manpreet Kaur Rangian, Upasna Attri. Design and Implementation of Malware Collection System Based on Client Honeypot. International Journal of Scientific & Engineering Research, 2013.
[4]. Cuckoo Sandbox. Доступно по ссылке: https://cuckoosandbox.org/, дата обращения 17.11.2018.
[5]. Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. Security & Privacy, IEEE, 2007.
[6]. Malware Anti-Analysis Techniques and Ways to Bypass Them. Доступно по ссылке: https://resources.infosecinstitute.com/malware-anti-analysis-techniques-ways-bypass/, дата обращения 02.05.2017.
[7]. Tal Garfinkel, Mendel Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. Computer Science Department, Stanford University, 2003. Доступно по ссылке: https://suif.stanford.edu/papers/vmi-ndss03.pdf, дата обращения 17.11.2018.
[8]. Классификация вредоносных программ. Доступно по ссылке: https://www.kaspersky.ru/blog/klassifikaciya-vredonosnyx-programm/2200/.
[9]. What is ransomware?. Доступно по ссылке: https://us.norton.com/internetsecurity-malware-ransomware.html, дата обращения 17.11.2018.
[10]. Drakvuf. Доступно по ссылке: https://drakvuf.com/, дата обращения 17.11.2018.
[11]. Tamas Kristof Lengyel. Malware Collection and Analysis via Hardware Virtualization. University of Connecticut, 2015. Доступно по ссылке: https://tklengyel.com/thesis.pdf, дата обращения 17.11.2018.
[12]. Xen Project. Доступно по ссылке: https://xenproject.org/, дата обращения 17.11.2018.
[13]. LibVMI. Доступно по ссылке: http://libvmi.com/, дата обращения 17.11.2018.
[14]. Recall Forensics. Доступно по ссылке: http://www.rekall-forensic.com/, дата обращения 17.11.2018.
[15]. QEMU. Доступно по ссылке: https://www.qemu.org/, дата обращения 17.11.2018.
[16]. Rekall Profiles. Доступно по ссылке: http://blog.rekall-forensic.com/2014/02/rekall-profiles.html, дата обращения 17.11.2018.
[17]. М. Руссинович, Д. Соломон, А. Ионеску. Внутреннее устройство Microsoft Windows. 6-е издание. Основные подсистемы ОС. СПб.: Питер, 2014, 672 с.
[18]. Джеффри Рихтер, Кристоф Назар. Windows via C/C++. Программирование на языке Visual C++. СПб.: Питер, 2009, 896 с.
[19]. Building C/C++ Programs. Доступно по ссылке: https://docs.microsoft.com/en-us/cpp/build/building-c-cpp-programs?view=vs-2017, дата обращения 17.11.2018.
[20]. Specifying Device Types. Доступно по ссылке: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/specifying-device-types, дата обращения 17.11.2018.

Methodology and Tools for Development and Verification of formal fUML Models of Requirements and Architecture for Complex Software and Hardware Systems

A.V. Samonov <a.samonov@mail.ru>
G.N. Samonova <g.samonova@mail.ru>
Mozhaiskiy Military Space Academy,
13, Zhdanovskaya St., Saint Petersburg, 197088, Russia

Abstract. The article presents models and algorithms to support end-to-end quality control of complex software and hardware systems through the implementation of the software-controlled process of development and verification of formal models of requirements and architecture of such systems, Firstly, we give the analysis of scientific publications and the normative-methodical base in the field of development and application in practice of the model-based approach is given. We establish that least provided by model, algorithmic and software solutions are issues related to the development of a complete and correct set of requirements, as well as the formalization and verification of technical projects of software and hardware systems. To solve the existing problems, we propose to develop a special unified environment for the development, modeling and testing formal models of requirements and architecture of complex software and hardware systems. These models provide an optimal set of interconnected fUML diagrams presented in ALF notation and verified in the fUML virtual machine and using SMT/SAT solvers.

Keywords: activity diagrams; class diagrams; design and implementation; life cycle of automated systems; model of requirements; model of architecture; software and hardware systems; verification and validation

DOI: 10.15514/ISPRAS-2016-30(5)-8

For citation: Samonov A.V., Samonova G.N. Methodology and Tools for Development and Verification of formal fUML models of Requirements and Architecture for Complex Software and Hardware Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 123-146. DOI: 10.15514/ISPRAS-2018-30(5)-8

1. Introduction

Now, when the confrontation in the political, economic and military fields is growing, one of the most important activities of the state is to ensure the safe operation of critical information infrastructure (CII). According to the Federal Law of the Russian Federation [1], CII objects are automated control systems (ACS) for

production and technological processes of the critical objects of the Russian Federation and information and telecommunication networks providing them, IT systems and communication networks for solving public administration tasks, ensuring defense capability, security and law enforcement. Disruption of the functioning of CII objects can lead to disastrous consequences in the field of defense capability, economy, health care and security of the nation.

Automation means complexes, which form the basis of the CII objects, are complex software and hardware systems (CSHS); their foundation of reliable and safe functioning is laid in the process of their design, development, and verification. The main factors and conditions for achieving the required quality indicators of CSHS are:

1) implementation of a quality management system defined by modern normative-methodical documents (NMD) in the field of system and software engineering at companies developing CSHS;
2) highly qualified designers, developers, and testers of CSHS;
3) use of modern technologies, methods and tools for design, development, and testing of CSHS.

The most important issues relate to the implementation of the third direction, which is being developed in system and software engineering [2] and model-based methodology [3]. The need to improve the technology and development tools of CSHS is due to distressing statistics on the implementation of IT projects both in Russia and abroad. Thus, according to the research of The Standish Group, the analysis of the results of work on the creation of information systems showed that in the United States (over the past 15 years), only 20% of the projects were completed on time and according to the original budget. At the same time, 30% of the projects failed; 50% faced various problems: the total budget exceeded the initial one by 2 times on average; the terms increased by 1.5 times; less than 75% of the required functionality was implemented [4]. The development process of CSHS consists of three main stages: justification of requirements, design, and implementation, each of which, according to the methodology of the model-based approach, includes a verification procedure of the corresponding artifact. As the analysis showed, issues related to the automation of the processes of generating and verifying computer code created at the implementation stage have been solved quite successfully. At the same time, the stages of requirements formation and system architecture design require the participation of specialists in the field of system engineering and information technology and end users.

As the analysis showed, the main limiting factors in achieving qualitative improvements in solving these tasks are:

- absence of a rigorous mathematical model describing the processes of implementation and application of methods and tools of model-based systems engineering in the main stages of the life cycle of CSHS in a uniform model-language environment;

- objective complexity of the task of creating a formal presentation of system requirements based on their original informal representation;
- availability of a wide range of languages and tools proposed for building models of the analysis, architecture, and implementation of a system in the absence of clear and specific rules and recommendations for their application;
- lack adequate tools for automated construction and execution of test scenarios for the verification of requirements and architecture.

The second section provides a brief overview of scientific and technical publications, in which the described issues are considered and solved. The third and fourth sections of the article present the models and algorithms for building a formal specifications requirements. The fifth section describes the models and algorithms for developing and verification the architecture of CSHS. The sixth section presents the methodology for constructing test scenarios to verify models of requirements, architecture, and implementation of CSHS using the SAT/SMT solvers.

2. Overview of the Current Normative-methodical Base and Scientific Publications in the Field of Development and Verification of CSHS

The exceptional relevance of the problems described above has led to the great attention and efforts taken by international and national organizations, scientific and professional communities, development teams and individual researchers to solve them. In the authors' opinion, the most important ones are methodical documents and specifications developed under the auspices of the OMG (Object Management Group) organization that cooperates with about 800 research organizations (DISA, INCOSE, NIST, etc.) and industrial companies (AT & T, IBM, Oracle, Microsoft, Cisco Systems, NASA, etc.). In Russia, active research in this area is carried out by such organizations as ISP RAS, the Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University, Saint Petersburg State University, Novosibirsk State Technical University, Military Space Academy named after A. F. Mozhaisky, etc.

Currently, more than 230 methodical documents and specifications have been published on the OMG website. Considering the issues described above, the most important specifications are: MOF (Meta Object Facility), UML (Unified Modeling Language), XMI (XML Metadata Interchange), SysML (System Modeling Language), OCL (Object Constraint Language), UTP (UML Testing Profile), ALF (Action Language for Foundational UML), FUML (Semantics of a Foundational Subset for Executable UML Models), ReqIF (Requirements Interchange Format).

These documents are the scientific and methodical base for their application, further improvement, and development. A brief analysis of the most important scientific publications and papers starts with monographs and practical guidelines in the field of industrial development of CSHS.

The fundamental paper written by Dragan Milicev, the Serbian scientist and MBSE expert, Professor of University of Belgrade [5], outlines the principles and methods of applying modern information technologies based on the object-oriented paradigm and model-based approach for the industrial development of CSHS. This is especially valuable in the context of the problems considered in this article. Also, the paper provides recommendations and examples of using the fundamental UML (fUML) language, which is used to create and verify executable formal UML models.

In the monograph [6], the techniques and methods of applying the constructs and mechanisms of the SysML language are described in a summary and illustrated form containing practical examples, the idea and principles of this language are explained. This monograph is written by the group of active developers of many OMG methodical documents and specifications, and those who apply this knowledge in practice at such companies as Lockheed Martin and Raytheon Company: S. Friedenthal, A. Moore, R. Steiner. Useful information on applying the SysML language mechanisms for designing CSHS is presented in the monograph by Lenny Delligatti [7] (Lockheed Martin Corporation).

From among all publications of Russian organizations and researchers, it is worth to mention the papers by the ISP RAS team dealing with both theoretical and practical aspects of these problems. The theoretical foundations of the design and verification of CSHS based on a category-theoretic approach to metaprogramming are described in publications written by S. Kovalev, the leading ISP RAS researcher [8] [9]. They present the ways to apply category theory to solve the problem of representing heterogeneous software engineering technologies in a common format that would be convenient for their integration and coordination in the software system design life cycle. Particular attention is paid to such modern technologies as model checking development and aspect-oriented programming, for which universal category-theoretic semantic models are built.

One of the modern means to describe the architecture of software and hardware systems is Architecture Analysis & Design Language (AADL) [10]. On the basis of this language, the system for supporting the design and verification of MASIW onboard aircraft systems developed by ISP RAS together with GosNIIAS as part of the state program for the development of Integrated Modular Avionics (IMA) is being actively used. When developing MASIW, the following libraries and tools were used: Eclipse Modeling Framework, Graphical Editing Framework, Eclipse Team Providing, SVN Team Provider, GIT Team Provider. As noted in the article [11], the MASIW tools allow solving the following tasks:

- creation, editing, and management of models of hardware-software complexes (HSCs) using the AADL language;
- analysis of models for the sufficiency of hardware resources and interface consistency, the evaluation of the characteristics of projected data networks built in accordance with the AFDX standard (Avionics Full-Duplex Switched Ethernet);

- distribution of functional applications over computation modules, taking into account the limitations of the hardware platform resources and the requirements for the reliability and security of HSCs;
- generation of computer code and configuration data for VxWorks653 RT OS and termination units of the AFDX network.

An example of using the special extension of the AADL language – Error Model Annex (EMA) and the MASIW tool for modeling and analyzing the security of the designed HSCs is presented in [12]. The model is created using EMA, in which a finite-state machine (FSM) is developed for each component of HSCs. The states of FSM are normal states and emergencies, including dangerous and failure situations of this component. The effect of system component failures on other components is described by specifying the logical conditions for the propagation of errors between different types of components in different states, taking into account the probabilities of their occurrence. The following algorithms are used for risk analysis: Fault Tree Analysis, Failure Mode and Effects Analysis, Markov Analysis. The implementation of the approach described in this article helps to identify and eliminate the security-critical defects in design solutions at the design stage.

The ISP RAS team has developed the technology called UniTESK (Unified TEsting Specification based toolkit) for testing software interfaces. This is a unified set of testing tools based on specifications. UniTESK is unified due to the fact that the general testing methodology and general architecture can be used to test modules using almost all programming languages. Currently, there are the UniTESK implementations for C (CTESK), C ++ (C ++ TESK), Java (JavaTESK and Summer), Python (PyTESK). The UniTESK technology has two main differences from common testing tools [13]:

- UniTESK helps to describe the specifications of a software contract of modules in the form of pre- and post-conditions using the extensions of programming languages (in case of C ++ TESK, no extension is required);
- instead of manual development of test cases, UniTESK allows describing a generalized scenario – a compact description of test logic that allows the test sequence generator to call each specified interface in all its uses automatically and to verify the correctness of the result for compliance with a specified post-condition.

The next group of publications consists of papers devoted to the solution of particular problems of developing and verifying CSHS. The thesis written by A.V. Markov, the employee of Novosibirsk State Technical University, is devoted to the issues of automation of design and software analysis processes using the UML language and Petri nets [14]. The paper describes the software design methodology using UML sequence diagrams in the .xmi format and presents the method for their automatic convert to the .cpn format used to describe Petri nets. The result of using this method is hierarchical Petri nets being analyzed for verifying the software

project, which is represented in the form of UML diagrams. The following solutions presented in this paper are the most valuable in practice:

- algorithm of transforming UML diagrams to Petri network;
- algorithm and rules of implementing inversion in Petri nets to check the reachability of the selected network state;
- algorithms and software for constructing and analyzing Petri nets to identify and eliminate defects in the developed software.

The review of modern methods for automatic test generation presented in [15] is quite useful. The paper describes the following methods:

- structural testing using symbolic execution;
- model-based testing;
- combinatorial testing;
- random testing;
- search-based testing.

The article [16] presents the automated method for making UML sequence diagrams using the description of UML use case diagrams and class diagrams. To implement this method, it is necessary to use the ATL language and metamodels of use case diagrams, class diagrams and sequence diagrams developed by the authors of the article, as well as the rules for obtaining the third diagram from the first and second ones. The result of this transformation is a sequence diagram in the XMI format, which is then converted to the XSLT format to display a sequence diagram in a graphical editor for viewing, analysis, and making changes. The disadvantage of the proposed algorithm is the lack of automatic correction of the original models if any new changes are made to a sequence diagram. This is due to the fact that the transformations using the ATL language are unidirectional – they work with read-only source models and create write-only target models.

In the work [17], experts at Shanghai University have described the approach to verify large-scale web projects by developing and analyzing the executable model of the corresponding software. To build this executable model, the authors have developed the method that uses live sequence charts (LSCs) as input data. A UML model using LSCs diagrams is transformed into a symbolic finite-state machine. Test scenarios are created by traversing a finite-state machine with the Depth-first Search method (DFS).

The paper [18] describes the method of automatic generation of computer code based on the project (architectural model) of a program presented in the ALF language. Of particular interest is the conceptual scheme of the mechanism for generating computer code from the project description in the ALF format using the rules in the extended Backus-Naur (EBNF) notation). The authors point out the following advantages of the tool to transform the model of the architecture of the ATL language: the ability to describe both declarative and imperative language constructs, the presence of means to combine modules that allow creating and

reusing sets of transformation rules. The result is a Java code that corresponds to the Modisco Java metamodel.

The article [19] describes two methods for implementing automatic testing of real-time loaded systems using scenarios. In the first, the system is modeled as the network of timed automata (TA). In the second, it is modeled as a set of live sequence charts (LSCs) and requirements in the form of a separate LSC diagram to analyze. The authors of the article have developed temporal extensions for a subset of the core of the LSC language and defined its semantics based on tracing. The analyzed LSC diagram is transformed to its behavioral equivalent in the notation of the TA diagram. The correctness verification of a model is carried out by modeling the TA diagram in real time using Computational Tree Logic (CTL) followed by the comparison of the obtained result with the standard. Both methods are implemented with the tools of UPPAAL.

The paper [20] describes the method for generating unit cases based on the architecture of a model presented in the form of UML activity diagrams. The tests are created with the SMT/SAT solvers, which analyze the control flow graph of a program presented in A Modeling Language for Mathematical Programming (AMPL). This paper proposes test coverage criteria based on control flow analysis. Particular attention is paid to mixed integer nonlinear programming, as well as to the construction of logical formulas for OCL (Object Constraint Language) constraints.

One of the serious disadvantages of modern approaches is the lack of ability to take into account the composition and structure of designed systems, as well as to establish and synchronize the relations between system requirements and design elements. To eliminate these disadvantages, the paper [21] proposes to make a system design based on SysML behavioral diagrams. To verify automatically the project created in this way, it is proposed to use the following methods:

- transformation of SysML activity diagrams to modular Petri nets presented in PNML (Petri Net Markup Language);
- mathematics and such tools as CPN Tools and SPIN for analyzing Petri nets;
- algorithm for verifying the time requirements in SysML activity diagrams, which are pre-converted to formulas of Linear Temporal Logic (LTL) using Active Temporal Requirement Language (AcTRL) developed by the authors.

To create tools for the dynamic verification and validation of project behavioral models, it is proposed to use Executable Domain-specific Modeling Languages (xDSMLs) in [22]. Means based on them make it possible to monitor the states of analyzed models (transitions, events, variable values) during their execution. The new generative approach based on a multidimensional and domain-specific trace metamodel is proposed. This method helps to construct and manage execution traces for models corresponding to a specified xDSML. According to the authors of

this paper, this method has higher performance compared to the standard UML metamodel due to the ability to exclude redundant data from processing (for example, analyzed traces) using the mechanisms of the corresponding xDSML.

To conclude the analysis of publications and the solutions presented in them, the following ideas can be summarized:

- main efforts of researchers are aimed at developing methods and tools for the automated generation and verification of software implementations of CSHS [13] [14] [15] [17] [18] [20]; fewer efforts are aimed at automating the development and verification of design solutions [11] [19] [20] [22]; there are practically no solutions for the automated formation and verification of a set of requirements;
- mathematics and analysis of Petri nets, SMT/SAT solvers, such modeling languages as AADL, UML, fUML, SysML and domain-specific languages (xDSMLs) developed on their basis are used as the basic mathematical models and tools for automatic verification based on these models.

In this regard, the main purpose of research and papers, the results of which are presented in this article, was to develop a model, algorithmic and methodical support of the processes of building and verifying formal models of requirements and the architecture of CSHS used in state CII objects.

For create unified conceptual, language and instrumental environment for the development and verification of analysis models and the architecture, it is proposed to use:

- UML, OCL, fUML and ALF modeling languages;
- VM fUML, SPIN (Promela), Rodin (Event-B), SMT-Lib, Z3, CVC-4, Alt-ERGO;
- environment, libraries and software products implemented within the Eclipse project: Eclipse Modeling Framework, Graphical Editing Framework, Papyrus, Moka.

The choice of these models, languages and tools is conditioned by the following circumstances. First, their development is actively supported by leading development enterprises and consumer organizations of CSHS. The second is that both the technologies and means based on them are open and available for study, application, and improvement.

3. Models and Algorithms of Formal Description of the Requirements for a System Based on the Original Informal Representation

To solve the problem of building a formal description of the requirements for automated systems and software, you must perform the following operations and procedures:

1. First, additional content control elements are developed and installed in a text editor (MS Word or Writer). These elements are XML schemas (*tz_as.xsd, tz_sw.xsd.*) based on a universal Requiment Interchage Format (ReqIF). XML schemas describe the composition and structure of requirements for automated systems and software defined in the relevant normative-methodical documents.
2. Then in the environment of a text editor in accordance with the established in the previous step xml schemas (tz_as.xsd and tz_sw.xsd) structured text documents are developed containing requirements to the system.
3. The next step is the automatic generation of the first version of the formal model of the set of requirements. to implement this procedure, use the metamodel shown in the Fig. 1. This metamodel is a conceptual and logical union of a use case diagram and a class diagram.

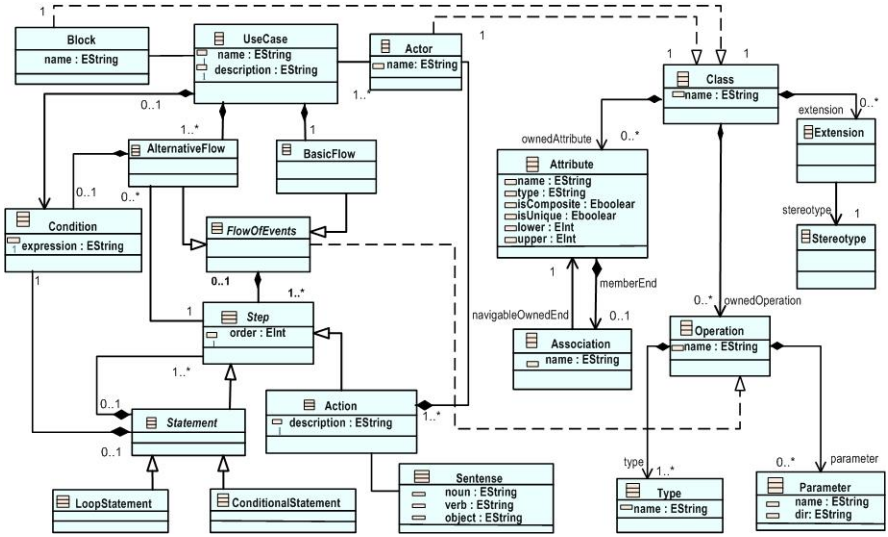

Fig.1. Comprehensive model of the use case diagram and class diagram

To develop this metamodel, the official specifications of these diagrams on the OMG website and the models proposed in publications [16] [22] [23] were used. In addition to the explicit establishment of relationships between diagram elements of these diagrams, the proposed model includes the new class – "Sentence" and excludes two classes – "Subject" and "Agent". The program implementing the generation procedure uses the *xmi* representation of this metamodel and developed before structured text documents containing requirements to the system.

Each *i*-th use case is a functional requirement and is described as follows:

$uc_i = (nameuc_i, actor_k, function_t, block_j)$,

where $nameuc_i$ – use case name uc_i;

$actor_k$ – user or external system that initiates uc_i;

$function_i$ – system function that implements uc_i;

$block_j$ – system component that implements $function_i(input, operation, result)$,

where $input$ input – input data;

$operation$ operation – algorithm that implements $function_i$;

$result$ result – result of the implementation of $function_i$.

$function_i = (basicflow, alterflow)$,

where

$basicflow$ – algorithm that implements the main flow of the function;

$altersflow$ – algorithms that implements alternative flows of the function.

The *class* construction is developed for each functional block (module) and information object. Its attributes, operations (methods), restrictions and semantics are specified. The sets of interacting classes are combined into class diagrams – d_class. Formally, a class diagram can be described as follows:

$d\_class = (Classes, Relations)$,

where $Classes = \{class_i\}\ i = 1, \dots, I$ – diagram classes;

$Relations$ – class relationships;

$class = (nameclass, attributes, operations)$,

$Relations = \{R_{as}, R_{in}, R_{ag}, R_{de}, R_{sp}, R_{re}\}$ – relations between classes of the following six types;

R_{as} – associations;

R_{in} – inheritances;

R_{ag} – aggregations;

R_{de} – dependences;

R_{sp} – specializations;

R_{re} – realizations.

The next step in building a requirements model is to develop non-functional requirements specifications for each system function:

$d\_reqs = \left(r_1^{f_i}, r_2^{f_i}, r_3^{f_i}, r_4^{f_i} \dots\right)$,

where $r_1^{f_i}$ – requirements for the efficiency of execution of f_i;

$r_2^{f_i}$ r_2^{fi} – performance requirements (for example, the amount of data stored, processed and transmitted, the number of users, the number and size of requests per unit of time, etc.);

$r_3^{f_i}$ $\mathrm{r}_3^{\mathrm{fi}}$ – requirements for reliability (availability rate, uptime, recovery time, etc.);

$r_4^{f_i}$ – security requirements.

The model built in this way is preliminary, and it is used as input data for the algorithm for building a model formal requirements in the fUML language which described in the next section.

4. Algorithm for Building a Formal Model Requirements

The scheme of the algorithm that implements the second stage of the procedure of building a formal requirements model using the fUML language is shown in Fig. 2. Use case diagrams (UCDs) – *d_uc*, class diagram (CDs) – *d_class* and requirements diagrams (RDs) – *d_reqs* are used as initial data. The analyst and future system user develop an interaction overview diagram (IOD) – *d_io* for each UCD (*d_uc*). In this diagram, the functions implemented by the system are described from the user's point of view in more detail using activity diagrams, sequence diagrams, and statechart diagrams. Formally, an interaction overview diagram can be represented as follows: $d\_io_i = \left(actor_k, io_i^{fi}, block_j\right)$, where $io_i^{fi} = (\mathrm{io}_\mathrm{i}^{\mathrm{fi}} = \left(\mathrm{a_k}, \mathrm{f_i}, \mathrm{b_j}\right)$ io_i^{fim}, io_i^{fia}) describes the algorithm for implementing the function by the *j*-th block (class) of the designed software, which includes the description of main (io_i^{fim}) and alternative (io_i^{fia}) flows.

Alternate flows describe the operation of programs in case of abnormal situations, such as erroneous user actions, unexpected influences from the external environment, etc. The main and alternative flows can have subordinate flows, which are described in IOD using frames with “ref”. The subordinate IOD flows show the work of a program from the user's point of view and can be represented with activity diagrams, sequence diagrams or statechart diagrams depending on the features of the functioning of CSHS and ways of the interaction with the user and environment. To describe the procedure and possibility of realization of those or other threads are used pre - and post- condition.

the model of requirements constructed in this way should be subjected to validation and verification procedures. The validation procedure is to assess the completeness and correctness of the set of requirements. It is carried out both by software tools and by the informal expertise of specialists in a particular subject area. Such properties of a model as consistency, systematicity, non-redundancy, security, liveliness, absence of deadlocks, impossible operations, looping are checked during verification. The verification of the requirements model is carried out through its execution and testing in the *fUML* virtual machine environment and analysis using *SAT/SMT solvers*.

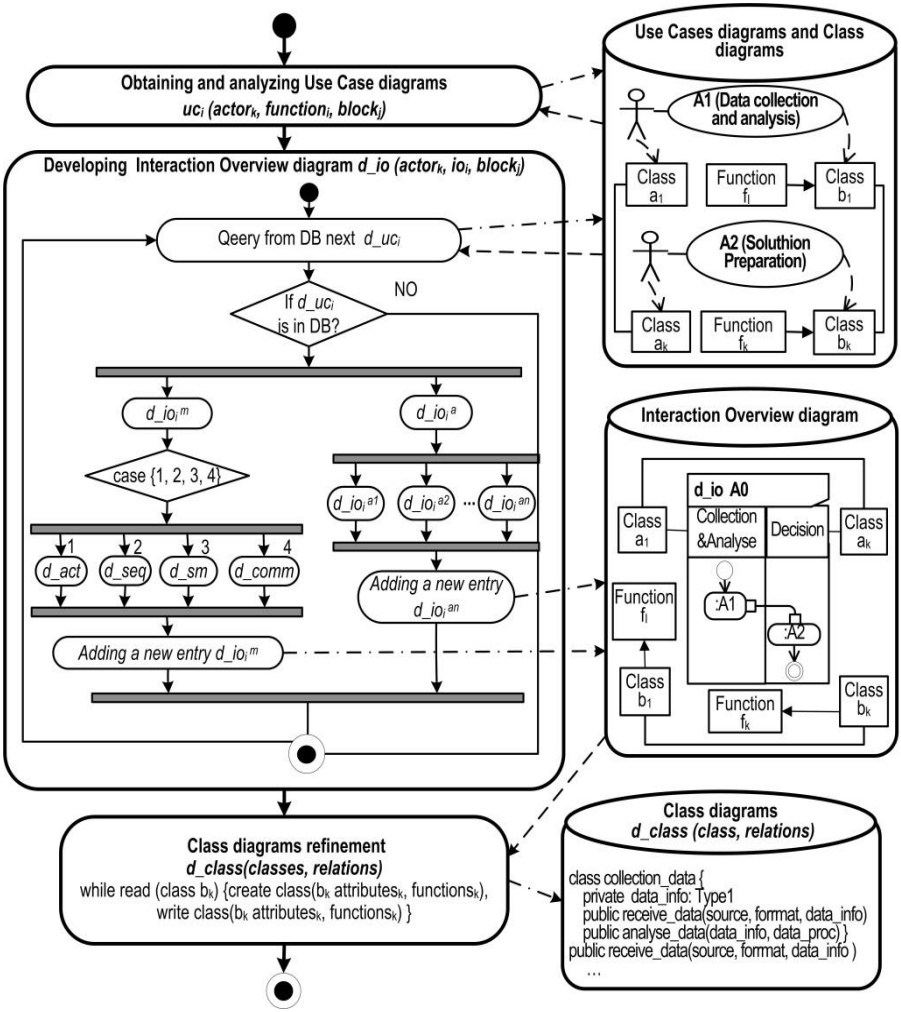

Fig.2. Construction algorithm of the technical project model

The description of these methods and tools is provided in sections 5 and 6.

5. Algorithm for Building a formal model of architecture of CSHS

The architecture development of CSHS is implemented in accordance with the algorithm shown in Fig. 3

The initial data are the interaction overview diagram – *d_io*, diagrams of quality requirements for the implementation of functions – *d_reqs*, class diagrams – *d_class* and the requirements for development technologies and operating environment.

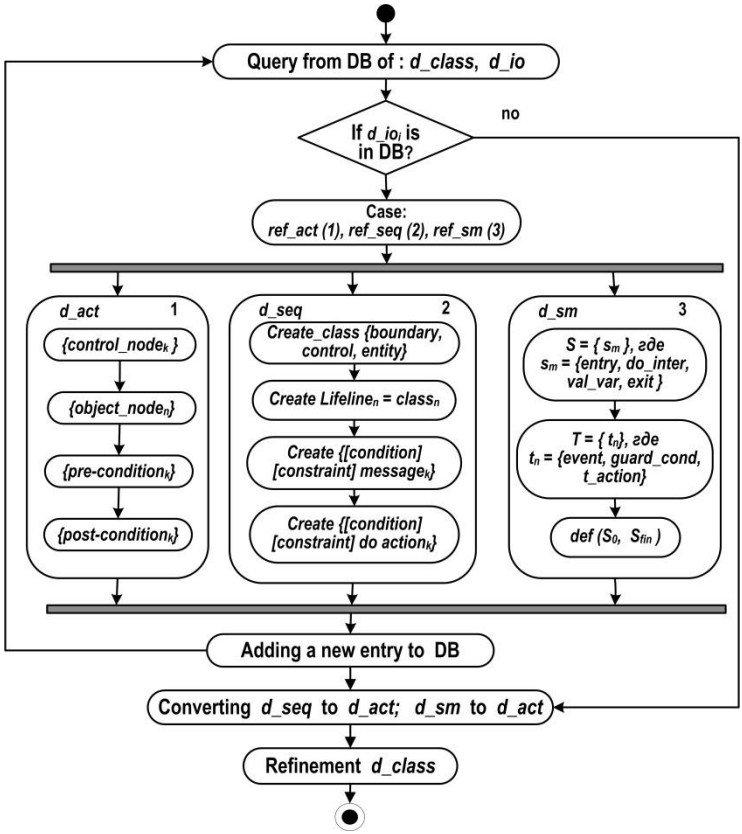

Fig.3. Algorithm of the architecture model development

In each interaction overview diagram (*d_io*) searches for a reference to activity diagrams (*ref_act*), sequence diagrams (*ref_seq*) and statechart diagram (*ref_sm*). If such references are found, the architect is asked to build or modify the corresponding diagrams. Activity diagrams are described using *control nodes* (*control_node: decision node, merge node, fork node, join node, interaction, interaction use); object nodes (object_node*); *pre-condition*s and *post-conditions*.

When constructing sequence diagrams, the additional *boundary*, *control*, and *entity* classes are first created, which perform the functions of intermediate (boundary) classes, control, and information objects, respectively. Then the *lifelines* are defined corresponding to classes that exchange *messages*. Messages are defined by the

conditions and limitations of their transmission and reception, and the actions that are performed (*do action*). When developing state diagrams, the $S = \{sm\}$ states and $T = \{tn\}$ transitions between them are defined. Each sm state consists of a description of the attributes - *val_var*, as well as the actions performed: *entry* – at the entrance, *do_inter* – internal, *exit* – at the exit.

The tn transitions include descriptions of the event initiating this transition – *event*, the pre- and post- implementation conditions – *guard_cond* and actions that must be performed before the actions of a new state – *t_action*).

The constructed diagrams are added to the database. To obtain a consistent and bound set of CSHS technical project (architecture) diagrams, class diagrams (*d_class*) and requirements diagrams (*d_regs*) are refined by establishing relations with new activity, sequences and statechart diagrams that were developed or modified. Fig. 4 shows the diagram illustrating the relationships between class and activity diagrams. Each d_act_i has a relationship with a specific class by describing the algorithm for implementing the corresponding class *method*.

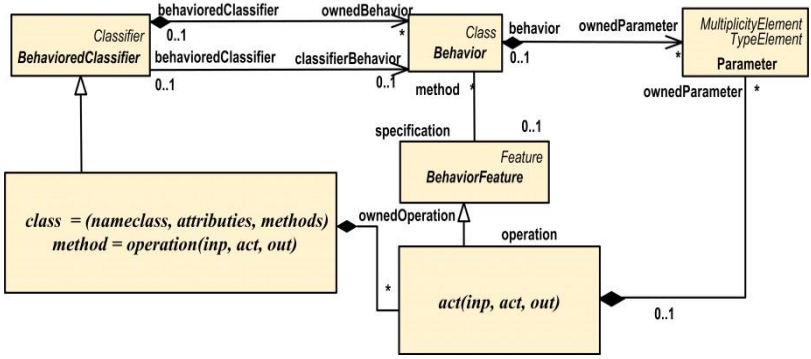

Fig. 4. Relationships and dependencies between the main components of the technical project model: class and activity

To implement the architecture model verification procedure in the virtual machine environment, *fUML* sequence diagrams (*d_seq*) and statechart diagrams (*d_sm*) are transformed to activity diagrams (*d_act*), which are then described in the language *ALF (Action Language for Foundational UML)*.

Fig. 5 presents the diagram illustrating the verification procedure of the formal *fUML* model of the CSHS architecture in a virtual machine environment consisting of three components: *ExecutionFactory*, *Executor* and *Locus*.

ExecutionFactory is used to create instances of the *visitor* semantic classes corresponding to the executable elements of the *fUML* model. The *Executor* class is a top-level abstraction for the executable *fUML* model and provides three operations:

- *evaluate* – evaluate a value specification, returning the specified value

- *execute* – synchronously execute a behavior, given values for its input parameters and returning values for its output behaviors;
- *start* – asynchronously start the execution of a stand-alone or classifier behavior, returning a reference to the instance of the executing behavior or of the behaviored classifier.

Each execution is performed on a specific VM (*Locus*), which is the abstraction of a physical or virtual computer capable of executing and verifying *fUML* models.

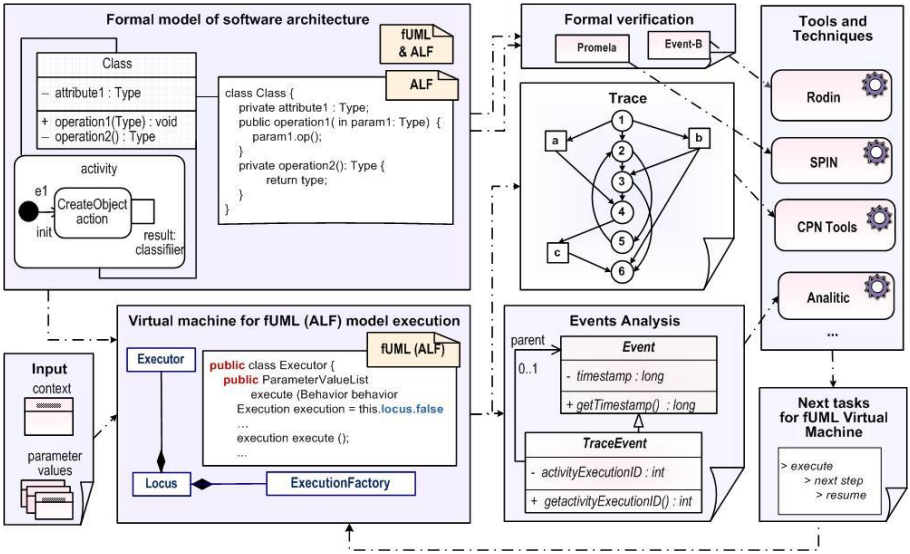

Fig.5. Scheme of executable fUML-model verification

The following basic requirements are imposed on the software architecture of CSHS:

- completeness of the implementation of functional requirements defined in the interaction overview diagrams – *d_io*;
- completeness and correctness of the implementation of non-functional requirements defined in requirements diagrams – *d_reqs*;
- coherence and consistency of all model diagrams;
- lack of redundancy.

Testing the architecture model in the *fUML* virtual machine environment also makes it possible to detect defects that can lead to security and liveliness violations, the occurrence of deadlocks, impracticable operations, and loops. In addition, it is

advisable to use SAT/SMT solvers to verify the architecture model. The description of their application is presented in the next section of the article.

6. Methods for Constructing Test Scenarios to Verify Models of Requirements, Architecture and Implementation of CSHS using SAT/SMT solvers

The main stages of the process of constructing test scenarios to verify models of requirements, architecture, and implementation of CSHS are presented in Fig. 6:

– building a control flow graph (CFG);

– description of CFG in language ALF;

– generation of test scenarios (TSs) for verification of a set of requirements and technical project (architecture);

– generation of TSs for implementation verification;

– adding test scenarios to database (DB).

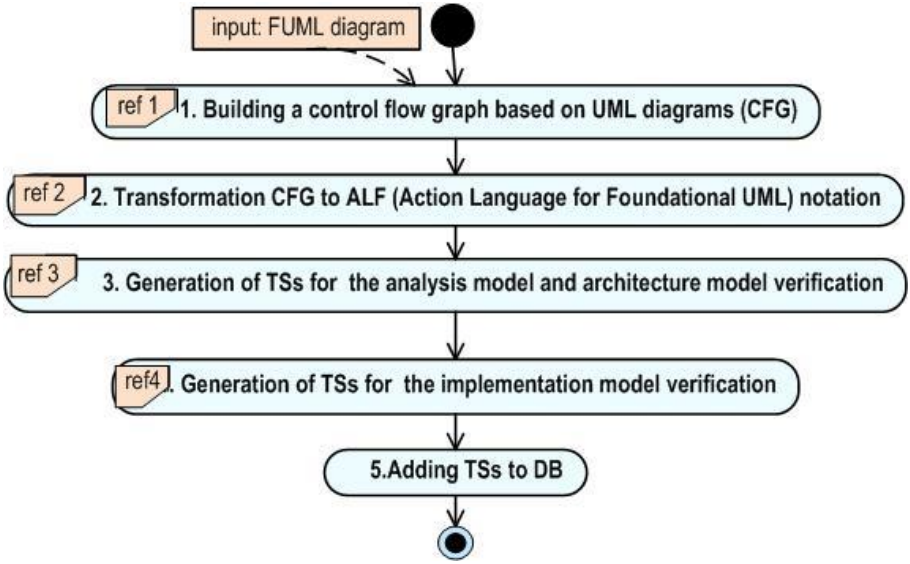

Fig.6. Generalized algorithm of test scenarios development for verification of requirement, architecture and implementation models

With the help of this algorithm, the requirements model and the architecture model can be verified. The original format for representing these models is *.xmi*. Based on these descriptions, the corresponding verifiable CFG model is built, in which both functional and non-functional requirements for the system being developed are taken into account. To represent non-functional requirements, *Object Constraint*

Language (OCL) is used. A SMT/SAT solver checks CFG for defects and, if they are found, creates counterexamples. Using them, the developer determines the causes of defects and makes the necessary corrections to the analyzed artifact. To implement this approach, it is proposed to use the ALT-ERGO, CVC4 and Z3 solvers, integrated into the Frama-C framework [24] [25].

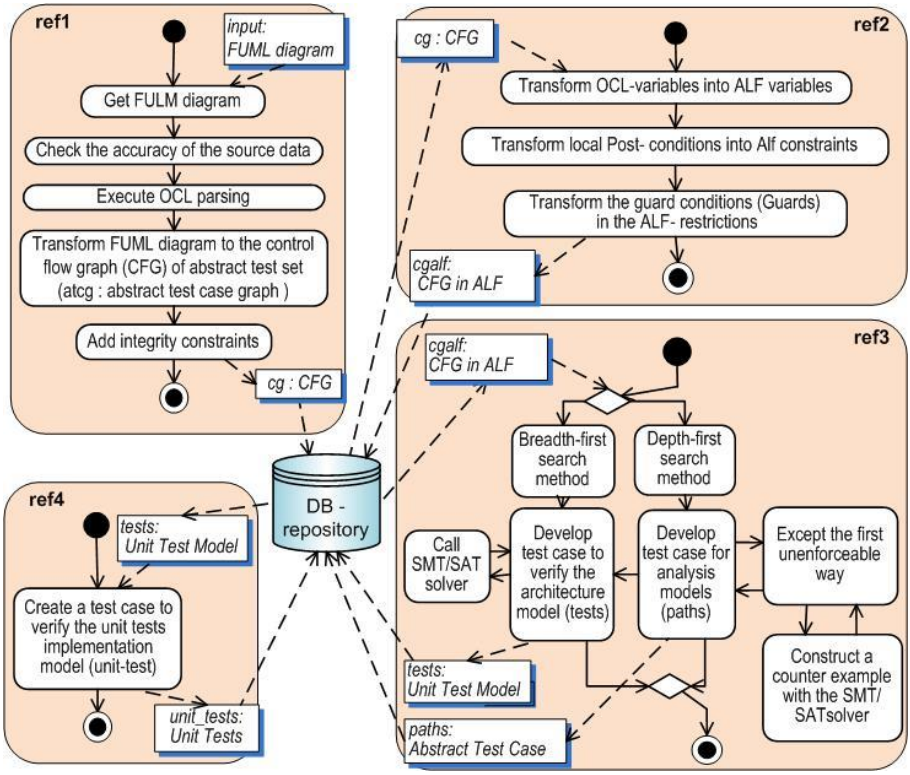

Fig. 7. Detailed algorithm of test scenarios development for verification models of requirement, architecture and implementation

Concluding the presentation of the developed models and algorithms, let us present a generalized scheme for the implementation of software-controlled process of development and verification of formal models of requirements and architecture of CSHS, which provides end-to-end quality control of all artifacts of the life cycle of CSHS (Fig. 8).

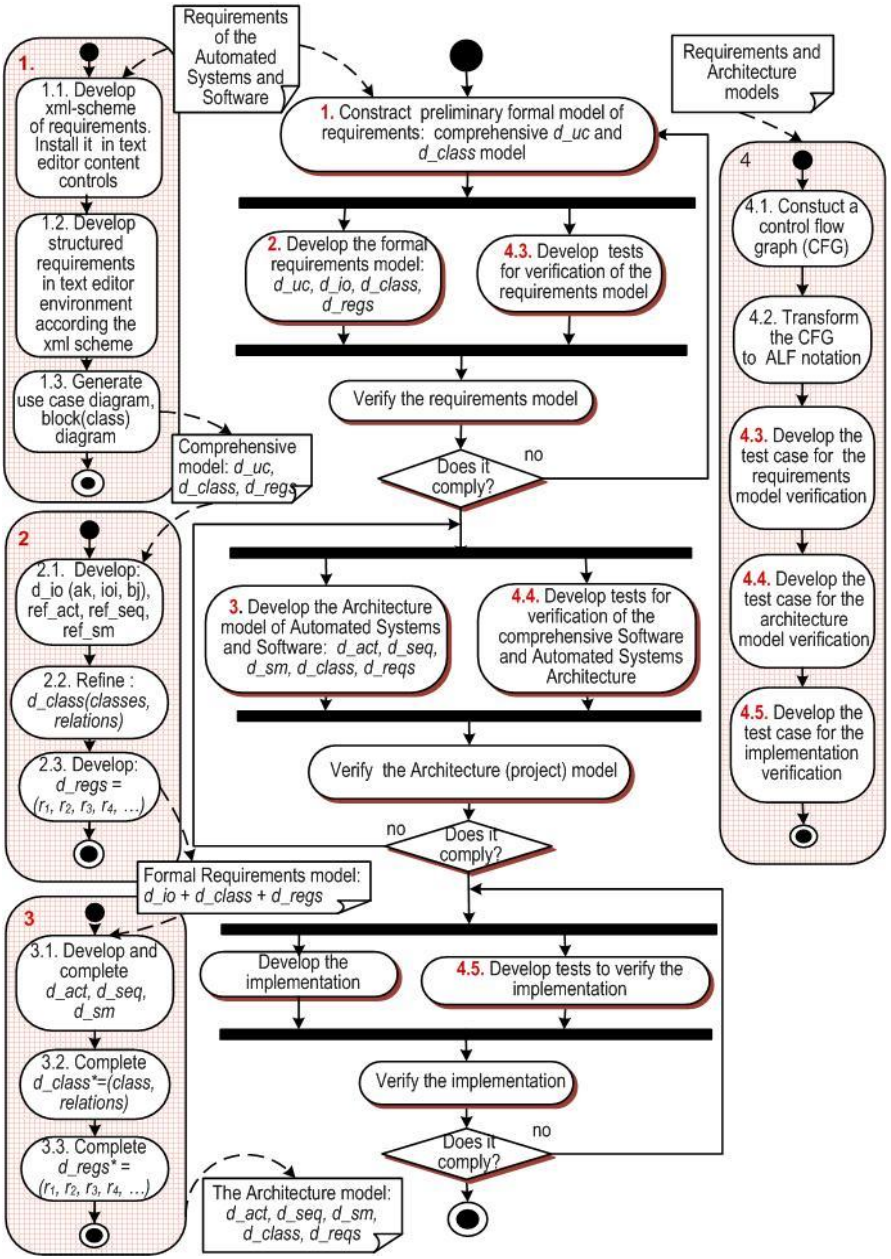

Fig.8. Stages of implementation of software-controlled of the process of development and verification of software and hardware systems

The main stages of the implementation of this approach are:

1. Construction of a preliminary formal model of requirements for CSHS in the form of a set of interrelated use case diagrams, class diagrams, and requirements diagrams.
2. Development of a formal requirements model in the form of a set of interrelated use case diagrams, overview interaction diagrams, class diagrams, and requirements diagrams.
3. Development and verification of the formal model of the architecture of CSHS through testing in the *fUML* virtual machine environment and analysis using SAT/SMT solvers – ALT-ERGO, CVC4 and Z3.
4. Development and verification of the software implementation.

7. Conclusion

One of the most important directions of improving the development processes and achieving the required quality indicators of complex software and hardware systems is the creation and implementation in practice of their industrial development of model-based technologies for justifying requirements, design, and implementation followed by the procedures of their formal verification and semi-formal validation. Currently, the most problematic issues are related to the verification of requirements and the CSHS architecture. To solve these problems, it is proposed to implement the approach described in this article. The distinctive features of this approach are:

- formation and use of a single model-language and information-software environment for the development and verification of formal models of requirements, architecture and software implementation based on the necessary and sufficient set of interrelated fUML diagrams and the model of internal and inter-model relations developed for them;
- implementation of the software-controlled development process of CSHS in accordance with the developed algorithm that performs sequential-iterative operations of generating and transforming formal models of requirements and architecture presented in *fUML*, *XMI*, *ALF*, and that also performs their verification in the fUML virtual machine environment and SMT/SAT solvers.

To implement the proposed approach, the following models, algorithms, and methods were developed:

- algorithm of a formal description of the requirements for the developed system based on the initial informal representation;
- fUML diagram models that are necessary and sufficient to develop complete, correct and consistent formal models of requirements and architecture;
- models, algorithms and guidelines for the development of formal models requirements and the architecture in languages fUML, XMI and ALF;

- verification algorithms for models of requirements and the architecture of CSHS in the environment of fUML virtual machine;
- verification of the formal model of the architecture and program implementation through the analysis using the SAT/SMT solvers.

Currently, work is underway to create a set of software tools to ensure the implementation of this approach. The development tools, libraries, and applications implemented in the Eclipse project (EMF, GMP, RCP, Papyrus, Moka, Titan) are used as a development environment and prototypes. The implementation of this software package in the relevant technological processes at companies will ensure the most complete accounting and correct implementation of requirements for functional and operational characteristics, environment and conditions for the use of CSHS. It will also significantly reduce the cost of finding and eliminating the most critical and resource-intensive defects made at the stages of the formation of requirements and design of their architecture.

References

[1]. Federal law "On security of critical information infrastructure of the Russian Federation". 12.07.2017 (in Russian)
[2]. Systems Engineering and Software Engineering, https://www.sebokwiki.org/wiki/Systems_Engineering_and_Software_Engineering. (accessed 25.07.2018).
[3]. Laura. Introduction To Model-Based System Engineering (MBSE) and SysML.https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf. (accessed 21.06.2018).
[4]. The Standish Group Report CHAOS. https://www.projectsmart.co.uk/white-papers/chaos-report.pdf. (accessed 25.08.2018).
[5]. Dragan Milicev. Model-Driven Development with Executable UML. John Wiley & Sons, 2009, 720 p.
[6]. Sanford Friedenthal, Alan Moore, Rick Steiner. A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann, 3 edition, 2014, 630 p.
[7]. Lenny Delligatti. SysML Distilled: A Brief Guide to the Systems Modeling Language. Addison-Wesley Professional, 2013, 304 p.
[8]. Kovalev S.P. Theoretical and categorical approach to metaprogramming. M., IPU Russian Academy of Sciences, 2014, 112 p. (in Russian)
[9]. Kovalev S.P. Category-Theoretic Approach to Software Systems Design. Journal of Mathematical Sciences, vol. 214, issue 6, 2016, pp. 814–853.
[10]. Peter H. Feiler, David P. Gluch. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 2012, 480 p.
[11]. D.V., Buzdalov, S.V. Zelenov, E.V. Kornykhin, A.K. Petrenko, V.A. Fear, A.A. Ognenko, A.V. Khoroshilov. Design tools for integrated modular avionics systems. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 201-230. DOI: 10.15514/ISPRAS-2014-26(1)-6 (in Russian)

[12]. S.V. Zelenov, S.A. Zelenova, Modeling of hardware and software systems and analyze their security. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 257-282. DOI: 10.15514/ISPRAS-2017-29(5)-13 (in Russian)
[13]. http://www.ispras.ru/technologies/unitesk (accessed 17.10.2018) (in Russian)
[14]. Markov, A.V., automation of design and analysis software using UML and Petri nets. PhD Thesis, NSTU, Novosibirsk, 2015 (in Russian).
[15]. Saswat Anand et al. An Orchestrated Survey on Automated Software Test Case Generation. Journal of Systems and Software, vol. 86, Issue 8, 2013, pp. 1978-2001.
[16]. Yachai Limpiyakorn, Photchana Sawprakhon. Sequence Diagram Generation with Model Transformation Technology. In Proc. of the International MultiConference of Engineers and Computer Scientists, IMECS 2014, vol I
[17]. Liping Li, Honghao Gao, Tang Shan. An Executable Model and Testing for Web Software based on Live Sequence Charts. In Proc. of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).
[18]. Thomas Buchmann and Alexander Rimer. Unifying Modeling and Programming with ALF. The Second International Conference on Advances and Trends in Software Engineering, vol I, IARIA, 2016. pp .10-15.
[19]. Shuhao Li, Sandie Balaguer et al. Scenario-based verification of real-time systems using Uppaal. Formal Methods in System Design, vol. 37, Issue 2–3, 2010, pp 200–264
[20]. Felix Kurth. Automated Generation of Unit Tests from UML Activity Diagrams using the AMPL Interface for Constraint Solvers. Master Thesis, Hamburg University of Technology (TUHH), 2014.
[21]. Messaoud Rahim, Malika Boukala-Ioualalen, Ahmed Hammad. Petri Nets Based Approach for Modular Verification of SysML Requirements on Activity Diagrams. PNSE'14, a satellite event of Petri Nets 2014 and ACSD 2014, Tunis, Tunisia, pp 233-248.
[22]. Erwan Bousse, Tanja Mayerhofer, Benoit Combemale, Benoit Baudry. Advanced and efficient execution trace management for executable domain-specific modeling languages. Software & Systems Modeling, 2017, https://link.springer.com/article/10.1007/s10270-017-0598-5 (accessed 20.07.2018)
[23]. D. Savic, S. Vlajic, S. Lazarevic. Use Case specification using the SilabReq domain specific language. Computing and Informatics, vol. 34, 2015, 877–910.
[24]. Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program Synthesis using Conflict-Driven Learning. In Proc.b of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18). ACM, New York, NY, USA, 16 p.
[25]. Efremov D. V., Mandrykin M. U. Formal verification of Linux kernel library functions. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 49-76. DOI: 10.15514/ISPRAS-2017-29(6)-3 (in Russian)

Методика и средства разработки и верификации формальных fUML моделей требований и архитектуры сложных программно-технических систем

А.В.Самонов <a.samonov@mail.ru>
Г.Н.Самонова <g.samonova@mail.ru>
Военно-космическая академия имени А.Ф. Можайского,
197088, Россия, Санкт-Петербург, ул. Ждановская, д.13

Аннотация. В статье представлены модели и алгоритмы обеспечения сквозного контроля качества сложных программно-технических систем (СПТС) посредством реализации программно-управляемого процесса разработки и верификации формальных моделей требований и архитектуры СПТС. Дан анализ научных публикаций и нормативно-методической базы в области разработки и применения на практике модельно-ориентированного подхода. Установлено, что наименее обеспеченными модельными, алгоритмическими и программными решениями являются вопросы, связанные с разработкой полного и корректного набора требований, а также с формализацией и верификацией технических проектов СПТС. Предложены способы решения существующих проблем посредством формирования единой модельно-языковой и информационно-программной среды разработки и верификации формальных моделей требований и архитектуры СПТС, построенных на основе оптимального набора взаимосвязанных fUML диаграмм, представленных в нотации языка ALF и верифицируемых в среде виртуальной машины fUML и с помощью SAT/SMT решателей.

Ключевые слова: верификация и валидация; диаграммы активности; диаграммы классов; жизненный цикл автоматизированных систем; модели архитектуры; модели требований; проектирование и реализация; программно-технические системы.

DOI: 10.15514/ISPRAS-2018-30(5)-8

Для цитирования: Самонов А.В., Самонова Г.Н. Методика и средства разработки и верификации формальных fUML моделей требований и архитектуры сложных программно-технических систем. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 123-146 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-8

Список литературы

[1]. Федеральный закон «О безопасности критической информационной инфраструктуры Российской Федерации». 12.07.2017 г.
[2]. Systems Engineering and Software Engineering https://www.sebokwiki.org/wiki/Systems_Engineering_and_Software_Engineering. (дата обращения 25.07.2018).
[3]. Laura E. Hart. Introduction to Model-Based System Engineering (MBSE) and SysML https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf. (дата обращения 21.06.2018).
[4]. The Standish Group Report CHAOS. https://www.projectsmart.co.uk/white-papers/chaos-report.pdf. (дата обращения 25.08.2018).

[5]. Dragan Milicev. Model-Driven Development with Ex ecutable UML. John Wiley & Sons, 2009, 720 p.
[6]. Sanford Friedenthal, Alan Moore, Rick Steiner. A Practical Guide to SysML: The Systems Modeling Language. Morgan Kaufmann, 3 edition, 2014, 630 p.
[7]. Lenny Delligatti. SysML Distilled: A Brief Guide to the Systems Modeling Language. Addison-Wesley Professional, 2013, 304 p.
[8]. Ковалёв С.П. Теоретико-категорный подход к метапрограммированию. М., ИПУ РАН, 2014, 112 стр.
[9]. Ковалев С. П. Теоретико-категорный подход к проектированию программных систем. Фундаментальная и прикладная. математика, том 19, вып. 3, 2014, стр. 111–170.
[10]. Peter H. Feiler, David P. Gluch. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 2012, 480 p.
[11]. Д.В. Буздалов, С.В. Зеленов, Е.В. Корныхин, А.К. Петренко, А.В. Страх, А.А. Угненко, А.В. Хорошилов. Инструментальные средства проектирования систем интегрированной модульной авионики. Труды ИСП РАН, том 26, вып. 1, 2014, стр. 201-230. DOI: 10.15514/ISPRAS-2014-26(1)-6
[12]. Зеленов С.В., Зеленова С.А. Моделирование программно-аппаратных систем и анализ их безопасности. Труды ИСП РАН, том 29, вып. 5, 2017 г., стр. 257-282. DOI: 10.15514/ISPRAS-2017-29(5)-13
[13]. http://www.ispras.ru/technologies/unitesk (дата обращения 17.10.2018)
[14]. Марков А.В. Автоматизация проектирования и анализа программного обеспечения с использованием языка UML и сетей Петри. Канд. дис., Новосибирск, НГТУ, 2015.
[15]. Saswat Anand et al. An Orchestrated Survey on Automated Software Test Case Generation. Journal of Systems and Software, vol. 86, Issue 8, 2013, pp. 1978-2001.
[16]. Yachai Limpiyakorn, Photchana Sawprakhon. Sequence Diagram Generation with Model Transformation Technology. In Proc. of the International MultiConference of Engineers and Computer Scientists, IMECS 2014, vol I,
[17]. Liping Li, Honghao Gao, Tang Shan. An Executable Model and Testing for Web Software based on Live Sequence Charts. In Proc. of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).
[18]. Thomas Buchmann and Alexander Rimer. Unifying Modeling and Programming with ALF. The Second International Conference on Advances and Trends in Software Engineering, vol I, IARIA, 2016. pp .10-15.
[19]. Shuhao Li, Sandie Balaguer et al. Scenario-based verification of real-time systems using Uppaal. Formal Methods in System Design, vol. 37, Issue 2–3, 2010, pp 200–264
[20]. Felix Kurth. Automated Generation of Unit Tests from UML Activity Diagrams using the AMPL Interface for Constraint Solvers. Master Thesis, Hamburg University of Technology (TUHH), 2014.
[21]. Messaoud Rahim, Malika Boukala-Ioualalen, Ahmed Hammad. Petri Nets Based Approach for Modular Verification of SysML Requirements on Activity Diagrams. PNSE'14, a satellite event of Petri Nets 2014 and ACSD 2014, Tunis, Tunisia, pp 233-248.
[22]. Erwan Bousse, Tanja Mayerhofer, Benoit Combemale, Benoit Baudry. Advanced and efficient execution trace management for executable domain-specific modeling languages. Software & Systems Modeling, 2017,

https://link.springer.com/article/10.1007/s10270-017-0598-5 (дата обращения 20.07.2018)

[23]. D. Savic, S. Vlajic, S. Lazarevic. Use Case specification using the SilabReq domain specific language. Computing and Informatics, vol. 34, 2015, 877–910.

[24]. Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program Synthesis using Conflict-Driven Learning. In Proc. of 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'18). ACM, New York, NY, USA, 16 p.

[25]. Ефремов Д.В, Мандрыкин М.У. Формальная верификация библиотечных функций ядра Linux. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 49-76. DOI: 10.15514/ISPRAS-2017-29(6)-3

TLA+ based access control model specification

A.V. Kozachok <a.kazachok@academ.mks.rsnet.ru>
Academy of the Federal Guard Service,
35, Priborostroitelnaya St., Oryol, 302034, Russia

Abstract. The article describes TLA+ access control model specification for computer systems, ensuring compliance with the mandatory integrity and confidentiality monitoring requirements with considering memory-based covert channels. The distinctive feature of the model is taking into account the characteristics of the lifecycle of electronic documents and their operating procedure. To specify the access control model, Lamport's Temporal Logic of Actions language was chosen (TLA+). Its notation seems to be the closest to generally accepted mathematical notation and its expressive capabilities and tools allow describing and verifying systems specified as finite automata. The following actions are defined in the model: create/delete a subject, read, write, append (blind write), create/delete an object, grant/remove access rights, include an object, exclude a nested object, approve an object (document), archive an object (document), cancel an approved object (document), copy an object (document). The following invariants are also defined: the type invariant (includes checking the compliance of all fields of the object, the compliance of the subject type, the uniqueness of the subject and object identifiers) and the safety invariant (includes checking the confidentiality and integrity labels of the interacting subjects and objects, the correctness of rights assignment procedures).

Keywords: security models; computer systems; verification; modelling; temporal logic; security policy; access control.

DOI: 10.15514/ISPRAS-2018-30(5)-9

For citation: Kozachok A.V. TLA+ based access control model specification. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 147-162. DOI: 10.15514/ISPRAS-2018-30(5)-9

1. Introduction

The problems of ensuring information security become more acute when information technologies are developing and penetrating in all spheres of life. The complexity and amount of software being developed and used are constantly increasing, which leads to the emergence of new threats and vulnerabilities.

It should be noted that some vulnerabilities are caused not by typical errors when programming, but by errors when designing software systems in general. Such defects are quite difficult to detect and to correct during the operation phase.

One of the possible solutions to solve this problem is the modeling and verification of the algorithms being developed for compliance with specified properties.

It is especially important to model the protection mechanisms of computer systems. For example, "The Information Security Requirements for Operating Systems" by FSTEC of Russia and developed on the basis of these requirements and according to GOST R ISO/IEC 15408 protection profiles and security targets contain the requirements of the ADV_SPM.1 functional component to present a formal security policy [1, 2]. In scientific studies, the formal description of security policies and access control models in operating systems is also given special attention [3-6].

There are a number of formal languages and relevant software tools providing the ability of the formalized description of a mathematical model [7]: Alloy [8], B [9], Event-B [10, 11], VDM [12], Z [13], TLA + [14-17].

2. Problem Formulation

Access control systems in computer systems provide mechanisms for controlling and restricting access for users or processes (subjects) to a variety of objects.

As part of the research, the task was to develop a model for controlling access to computer system resources, which would ensure that the requirements of mandatory integrity and confidentiality monitoring are met, taking into account information flows by memory [18].

The distinctive feature of the model is taking into account the characteristics of the lifecycle of electronic documents and their operating procedure.

To specify the access control model, Lamport's Temporal Logic of Actions language was chosen (TLA+). Its notation seems to be the closest to generally accepted mathematical notation and its expressive capabilities and tools allow describing and verifying systems specified as finite automata [19-21]. Also, in some research works, this notation was used to solve the problem of verifying access control models [22, 23].

3. Model Specification

The extension of temporal logic [24] is Lamport's Temporal Logic of Actions. It allows describing interacting and open-loop systems.

Unlike predicate logic, temporal logic of actions has the following operators [14]:

□ "always in the future" operator;

■ "always in the past" operator;

◯ "next-time" operator;

⊖ "at one point in time" operator;

◊ "once in the future" operator;

♦ "once in the past" operator;

$\mathcal{U}$ "until" binary operator;

$\mathcal{S}$ "since" binary operator.

The basic relations between the operators can be represented as follows:

$$\Diamond F \equiv \neg\Box\neg F \qquad \blacklozenge F \equiv \neg\blacksquare\neg F$$

$$\Diamond F \equiv (F \vee \neg F)\mathcal{U}F \qquad \blacklozenge F \equiv F\mathcal{S}(F \vee \neg F)$$

Logical formulas in the proposed access control model are defined as follows (in the Backus-Naur form):

$$\langle\phi\rangle \mid= PredAction \mid p(t_1,\ldots,t_n) \mid \neg\phi$$
$$\mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \to \phi \mid \forall x : \phi$$
$$\mid \exists x : \phi \mid \Box\phi \mid \Diamond\phi \mid \bigcirc\phi \mid \phi\mathcal{U}\phi$$
$$\mid \blacksquare\phi \mid \blacklozenge\phi \mid \ominus\phi \mid \phi\mathcal{S}\phi,$$

where $PredAction$ – actions, p – arity of a predicate n, $t_1,\ldots,t_n$ – terms, x – variable.

In general, the specification of the access control model in TLA+ is as follows

$$Spec \triangleq Init \wedge \Box[Next]_{vars}, \qquad (1)$$

where $Init$ – initialization procedure of initial values of model variables, $Next$ – action predicate that changes the state of the model and the values of variables, $vars$ – variables of the model.

3.1 Definition of Model Variables

Variable values may change after the execution of action predicates:

$$\text{VARIABLES } A, O, S,$$
$$vars \triangleq \langle A, O, S\rangle.$$

where A – set of current (happened) access events, O – set of objects, S – set of subjects, $\triangleq$ – "equal by definition" symbol.

3.2 Creating Data Types Describing Objects and Subjects of the Model

TLA + does not have strong typing (only embedded types are checked by default); however, checking of invariants of types is an integral part of the specification, because the verification is performed by the ModelChecking method [25]. A number of values specified in the model are model for reducing the resource-intensiveness of the verification process, but they do not affect the generality and adequacy of the model as a whole.

Description of the type that specifies the objects:

$$Objects \triangleq [oid : ObjectIDs, meta : ObjectMeta, body : ObjectBody,$$
$$owner : SubjectIDs, grantm : GrantedRights,$$
$$grantb : GrantedRights, incl : ObjectIDs,$$
$$st : ObjectStates].$$

Description of the type that specifies the subjects:

$$Subjects \triangleq [sid : SubjectIDs, cnfl : ConfidLevels,$$
$$\text{int}\, l : IntegrLevels, cat : \text{SUBSET}\, Categories,$$
$$owner : SubjectIDs].$$

Sets of identifiers of subjects and objects (model values):

$$SubjectIDs \triangleq 0 \ldots 5,$$
$$ObjectIDs \triangleq 0 \ldots 5.$$

Sets of labels for the levels of confidentiality, integrity, and categories (model values):

$$ConfidLevels \triangleq 0 \ldots 1,$$
$$IntegrLevels \triangleq 0 \ldots 1,$$
$$Categories \triangleq \{"c1", "c2", "c3"\}.$$

Set of object states ("work", "approved", "archived", "cancelled"):

$$ObjectStates \triangleq \{"work", "approved", "archived", "cancelled"\}.$$

Set of types of access and tuple of rights assignment

$$Rights \triangleq \{"read", "write"\},$$
$$GrantedRights \triangleq \langle sid : SubjectIDs, r : Rights \rangle.$$

For electronic documents in electronic document management systems, there is the separation of rights of access to meta information and the content of a document [26]. This opportunity was also taken into account in the developed model:

$$ObjectParts \triangleq \{"meta", "body"\},$$
$$ObjectMeta \triangleq [cnfl : ConfidLevels, intl : IntegrLevels],$$
$$ObjectBody \triangleq [cnfl : ConfidLevels, intl : IntegrLevels].$$

Auxiliary operators and functions were also identified for the selection: child element of an object ($sc(o)$), set of child elements of an object ($scs(o)$), set of copies of an object ($scp(o)$), set of child objects of a subject ($sw(s)$) and update the owner of a subject ($UpdateOwner(sh, sp)$).

3.3 Initialization of Initial Values

The set of current access events is empty at the initialization stage. The set objects can also be empty at this stage. However, the set of subjects in this case must contain at least one subject. For example, the values of the sets of subjects and objects are initialized with model values:

$$
\begin{aligned}
s0 \triangleq\ & [sid \mapsto 0, cnfl \mapsto 1, intl \mapsto 1, \\
& cat \mapsto \{\text{"c1"}, \text{"c2"}\}, owner \mapsto 0], \\
s1 \triangleq\ & [sid \mapsto 1, cnfl \mapsto 1, intl \mapsto 0, \\
& cat \mapsto \{\text{"c2"}, \text{"c3"}\}, owner \mapsto 1], \\
o0 \triangleq\ & [oid \mapsto 0, \\
& meta \mapsto [cnfl \mapsto 0, intl \mapsto 0], \\
& body \mapsto [cnfl \mapsto 0, intl \mapsto 0], \\
& cat \mapsto \{\text{"c1"}, \text{"c2"}\}, \\
& owner \mapsto 1, \\
& grantm \mapsto \{\langle 0, \text{"write"}\rangle, \langle 0, \text{"read"}\rangle\}, \\
& grantb \mapsto \{\langle 0, \text{"read"}\rangle\}, \\
& incl \mapsto \{\}, \\
& copy \mapsto \{\}, \\
& st \mapsto \text{"work"}, \\
Init \triangleq\ & \wedge A = \{\} \\
& \wedge S = \{s0, s1\} \\
& \wedge O = \{o0\}.
\end{aligned}
\tag{2}
$$

where oid – object identifier, $meta$ – object meta information, $body$ – object content, $cnfl$ – level of confidentiality, $intl$ – level of integrity, cat – category, st – object state, $owner$ – owner, $incl$ – set of included documents, $grantm$ – access rights assigned to meta information, $grantb$ – access rights assigned to the content of an object, sid – subject identifier.

The model provides actions to create and delete subjects and objects; therefore, the values presented in (2) only allow modeling possible states and finding errors faster.

3.4 Predicates of Actions

The following possible actions are specified in the model:

$$\begin{aligned} Next \triangleq & \vee CreateSubjectD \vee DeleteSubjectD \\ & \vee ReadD \quad \vee CreateObjectD \\ & \vee WriteD \quad \vee AppendWD \\ & \vee DeleteObjectD \quad \vee GrantRightsD \\ & \vee RemoveRightsD \vee IncludeObjectD \\ & \vee ExludeObjectD \quad \vee ApproveObjectD \\ & \vee ArchiveObjectD \vee CancelObjectD \\ & \vee CopyObjectD, \end{aligned}$$

where *CreateSubjectD* – create a subject, *DeleteSubjectD* – delete a subject, *ReadD* – read, *WriteD* – write, *AppendWD* – write at the end ("blind" writing), *CreateObjectD* – create an object, *DeleteObjectD* – delete an object, *GrantRightsD* – assign access rights, *RemoveRightsD* – remove access rights, *IncludeObjectD* – include an object to an object, *ExludeObjectD* – exclude an included object, *ApproveObjectD* – approve an object (document), *ArchiveObjectD* – archive an object (document), *CancelObjectD* – cancel the action of an approved object (document), *CopyObjectD* – copy an object (document).

Taking the example of the action presented in (3), one can consider the order in which the necessary pre-conditions and post-conditions are specified. Pre-conditions are predicates, the execution of which is necessary to ensure the safety of an action. Post-conditions determine how model variables may be changed according to the results of an action.

The $Read(s, o, r, op)$ action is performed by s (subject) in relation to o (object) with the $r = "read"$ right and the specific component of the meta information document ($op = "meta"$) or its content ($op = "body"$). The action in the model defines the following post-conditions: the current access to the set of access events is added ($A' = A \cup \{\langle s.sid, o.oid, r, op\rangle\}$) and sets of subjects and objects remain as unchanged (UNCHANGED⟨S, O⟩).

The $ReadD$ action imposes requirements to account all possible states of the model ($\exists r \in R : \exists s \in S : \exists o \in O : \exists op \in ObjectParts$) and to verify the necessary safety predicates of the execution of the action (pre-conditions). $ReadD \triangleq \exists r \in Rights :$

$$
\begin{aligned}
&\exists s \in S : \\
&\exists o \in O : \\
&\exists op \in ObjectParts : \\
&\quad \wedge r = \text{"read"} \\
&\quad \wedge o.cat \subseteq s.cat \\
&\quad \wedge \vee \wedge op = \text{"meta"} \\
&\qquad \wedge s.cnfl \geq o.meta.cnfl \\
&\qquad \wedge \vee \{\langle s.sid, r \rangle\} \subseteq o.grantm \\
&\qquad\quad \vee o.owner = s.sid \\
&\quad\ \vee \wedge op = \text{"body"} \\
&\qquad \wedge s.cnfl \geq o.body.cnfl \\
&\qquad \wedge \vee \{\langle s.sid, r \rangle\} \subseteq o.grantb \\
&\qquad\quad \vee o.owner = s.sid \\
&\quad \wedge Read(s, o, r, op)
\end{aligned}
\tag{3}
$$

Important ones are checks of confidentiality labels ($s.cnfl \geq o.meta.cnfl$ and $s.cnfl \geq o.body.cnfl$), as well as checks of categories as a part of mandatory confidentiality monitoring ($o.cat \subseteq s.cat$). It then checks the condition if s (subject) has the r access right to o (object) depending on op ($\langle ssid, r \rangle \subseteq o.grantm$ or $\langle ssid, r \rangle \subseteq o.grantb$).

It is also possible that the subject s is the owner of the object o; in this case, the requirement to have a right in the set of access rights to the object is not imposed; the owner has full rights ($o.owner = s.sid$).

Consider also the action to create a copy of the $CopyObjectD$ object in (4).

$$
\begin{aligned}
&CopyObject(s, o, id) \triangleq \wedge O' = O \cup \{[oid \mapsto id,\\
&\qquad meta \mapsto [cnfl \mapsto o.meta.cnfl,\\
&\qquad intl \mapsto o.meta.intl],\\
&\qquad body \mapsto [cnfl \mapsto o.body.cnfl,\\
&\qquad intl \mapsto o.body.intl],\\
&\qquad owner \mapsto s.sid,\\
&\qquad grantm \mapsto o.grantm,\\
&\qquad grantb \mapsto o.grantb,\\
&\qquad cat \mapsto o.cat,\\
&\qquad incl \mapsto o.incl,\\
&\qquad st \mapsto \text{"approved"},\\
&\qquad copy \mapsto \{o.oid\}]\}\\
&\qquad \wedge A' = A \cup \{\langle s.sid, o.oid, \text{"copy"}, id\rangle\}\\
&\qquad \wedge \text{UNCHANGED}\langle S\rangle
\end{aligned}
$$

$$
\begin{aligned}
&CopyObject \triangleq \qquad (4)\\
&\quad \exists s \in S :\\
&\quad \wedge O \neq \{\}\\
&\quad \wedge \exists o \in O : \wedge o.owner = s.sid\\
&\qquad\qquad \wedge o.incl = \{\}\\
&\qquad\qquad \wedge o.st = \text{"approved"}\\
&\qquad\qquad \wedge s.cat \subseteq o.cat\\
&\qquad\qquad \wedge s.cnfl = o.meta.cnfl\\
&\qquad\qquad \wedge s.intl \geq o.meta.intl\\
&\qquad\qquad \wedge s.cnfl = o.body.cnfl\\
&\qquad\qquad \wedge s.intl \geq o.body.intl\\
&\qquad\qquad \wedge Cardinality(scp(o)) < 2\\
&\qquad\qquad \wedge \exists id \in ObjectIDs :\\
&\qquad\qquad\quad \wedge \forall oo \in O :\\
&\qquad\qquad\qquad \wedge id \neq oo.oid\\
&\qquad\qquad\qquad \wedge CopyObject(s, o, id)
\end{aligned}
$$

The $CopyObject(s, o, id)$ is performed by s (subject) in relation to o (object). As a parameter, the identifier is also passed for the object being created, the selection of which is carried out taking into account the requirement for the unique identifiers of all objects ($\exists id \in ObjectIDs : \forall oo \in O : id \neq oo.oid$). In this action, a new object is added to a set of objects possessing the attributes of the o original object with the exception of the *copy* field, in which it is indicated which object is a copy of this object. It also adds the current access to the set of access events and indicates that the set of subjects does not change when performing this action.

The $CopyObjectD$ imposes requirements to account all possible states of the model ($\exists s \in S : O \neq \{\} \wedge \exists o \in O$) and to check the necessary pre-conditions:

- the subject is the owner of the object ($o.owner = s.sid$);
- the object has no included objects ($o.incl = \{\}$);
- the object is in an "approved" state ($o.st = "approved"$);
- subject categories are a subset of object categories ($s.cat \subseteq o.cat$);
- the privacy level of the subject is equal to the privacy level of the object ($s.cnfl = o.meta.cnfl \wedge s.cnfl = o.body.cnfl$);
- the level of integrity of the subject is greater than or equal to the level of integrity of the object ($s.intl \geq o.meta.intl \wedge s.intl \geq o.body.intl$);
- the number of copies of the object does not exceed the specified value ($Cardinality(scp(o)) < 2$);
- the created object must have a unique identifier.

3.5 Model invariants

In addition to specifying the pre- and post-conditions in the model, it is possible to set invariants for global properties, which are mandatory for all states of the model. The basic and generally accepted invariant is an invariant of types in (5).

It verifies that all fields match all objects and also checks the conformity of the type of all subjects, as well as it checks the uniqueness of all identifiers of subjects and objects.

$$
\begin{aligned}
&ObjTypeInv \triangleq \\
&\qquad \wedge \forall o \in O : \wedge o.oid \in ObjectIDs \\
&\qquad \wedge o.meta \in ObjectMeta \\
&\qquad \wedge o.body \in ObjectBody \\
&\qquad \wedge o.owner \in SujectIDs \\
&\qquad \wedge \{o.incl\} \subseteq \text{SUBSET } ObjectIDs \\
&\qquad \wedge \{o.copy\} \subseteq \text{SUBSET } ObjectIDs \\
&\qquad \wedge o.st \in ObjectStates \\
&\qquad \wedge o.cat \subseteq \text{SUBSET } Categories \\
&TypeInv \triangleq \wedge S \subseteq Subjects \qquad (5) \\
&\qquad \wedge ObjTypeInv \\
&\qquad \wedge \forall sn \in S : \text{IF } \exists sm \in S : \wedge sm \neq sn \\
&\qquad\qquad\qquad\qquad \wedge sn.sid = sm.sid \\
&\qquad\qquad \text{THEN FALSE} \\
&\qquad\qquad \text{ELSE TRUE} \\
&\qquad \wedge \forall on \in O : \text{IF } \exists om \in O : \wedge om \neq on \\
&\qquad\qquad\qquad\qquad \wedge on.oid = om.oid \\
&\qquad\qquad \text{THEN FALSE} \\
&\qquad\qquad \text{ELSE TRUE}
\end{aligned}
$$

The second invariant specified in the model is the safety invariant in (6). It checks the following conditions:

- the level of confidentiality of object meta information does not exceed the level of confidentiality of the content of the object ($o.meta.cnfl \leq o.body.cnfl$);
- the integrity level of object meta information is equal to the integrity level of object content ($o.meta.intl = o.body.intl$);
- if the object contains an included object, then the set of access rights to the parent object is a subset of access rights to the child object, and the state of the parent object is equal to the state of the child one ($o.grantm \subseteq oi.grantm \wedge o.grantb \subseteq oi.grantb \wedge o.st = o.ist$);

- the number of included objects does not exceed the specified value ($Cardinality(scs(o)) \le 1$);
- the number of copies of the object does not exceed the specified value ($Cardinality(scp(o)) \le 2$);
- if the subject is the owner of the object, the rights for it are not assigned, because the subject has full rights ($\neg o.grantm \subseteq (s.sid \times Rights) \wedge \neg o.grantb \subseteq (s.sid \times Rights)$);
- if the object is in the "archived" or "canceled" state, then it is forbidden to assign the right to "record" to any of the subjects ($o.grantm \cap (SubjectIDs \times \{"write"\}) \neq \{\} \vee o.grantb \cap (SubjectIDs \times \{"write"\}) \neq \{\}$).

The execution of invariants for all states of the model provides the proof of the following theorem (7) regarding the specification of the model (1) and the invariants (5), (6):

Theorem. $Spec \Rightarrow \Box(TypeInv \wedge Safety)$. (7)

3.6 Model verification

The significant limitation of the approach to the verification based on the ModelChecking method is the need to check all possible model states. That is, if one specifies any conditions for countable sets, for example, $sid \in Nat$ or $oid \in Nat$, the verification process does not end, because the number of model states will also be countable. Therefore, in the specification, model values were used to reduce the time required for model verification.

The verification of the developed model was performed using the TLC2 tool version 2.13 [27]. Thus, the time spent on verification was about 2835 minutes (more than 47 hours) on the server with the Ubuntu 16.04 operating system, Intel Xeon E5-2620 v2 24 cores 2.10 GHz and 32 GB of RAM. 16,284,800,554 states were verified with the average system performance of 5,743,616 states per minute.

4. Conclusion

The developed model can be used to set policies for controlling access to computer system resources, where information of various confidentiality categories, as well as various levels of confidentiality and integrity, circulates. The TLA + language notation used in the model has sufficient flexibility and expressive capabilities for solving a wide range of modeling problems in computer security. We should note that the requirements for the absence of covert timing channels in this model were not taken into account; that is the direction for further research.

$$
\begin{aligned}
Safety \triangleq \wedge\ & \forall o \in O : \wedge\ o.meta.cnfl \leq o.body.cnfl \\
& \wedge\ o.meta.intl = o.body.intl \\
& \wedge\ \text{IF } o.incl \neq \{\} \\
& \quad \text{THEN } \forall i \in o.incl : \\
& \qquad \wedge\ \exists oi \in O : \\
& \qquad\quad \wedge\ oi.oid \neq o.oid \\
& \qquad\quad \wedge\ o.oid = i \\
& \qquad\quad \wedge\ o.grantm \subseteq oi.grantm \\
& \qquad\quad \wedge\ o.grantb \subseteq oi.grantb \\
& \qquad\quad \wedge\ o.st = oi.st \\
& \quad \text{ELSE TRUE} \\
& \wedge\ Cardinality(scs(o)) \leq 1 \\
& \wedge\ Cardinality(scp(o)) \leq 2 \\
& \wedge\ \exists s \in S : \\
& \quad \wedge\ o.owner = s.sid \\
& \quad \wedge\ \text{IF } o.grantm \neq \{\} \\
& \qquad \text{THEN } \neg o.grantm \subseteq (\{s.sid\} \times Rights) \\
& \qquad \text{ELSE TRUE} \\
& \quad \wedge\ \text{IF } o.grantb \neq \{\} \\
& \qquad \text{THEN } \neg o.grantb \subseteq (\{s.sid\} \times Rights) \\
& \qquad \text{ELSE TRUE} \\
& \wedge\ \neg \exists o \in O : \wedge \vee\ o.st = \text{"archived"} \\
& \qquad\qquad\qquad \vee\ o.st = \text{"cancelled"} \\
& \qquad\qquad\quad \wedge \vee\ o.grantm \cap (SubjectIDs \times \{\text{"write"}\}) \neq \{\} \\
& \qquad\qquad\quad \wedge \vee\ o.grantb \cap (SubjectIDs \times \{\text{"write"}\}) \neq \{\}
\end{aligned}
\tag{6}
$$

References

[1]. Devyanin P.N. On the problem of representation of the formal model of security policy for operating systems. Trudy ISP RAN/Proc. ISP RAS. vol. 29, issue 3, 2017, pp. 7-16 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-1.

[2]. Devyanin P. N. Approaches to formal modelling access control in postgresql within framework of the mrosl DP-mode. PDM. Prilozhenie/Applied Discrete Mathematics. Supplement, no. 10, 2017, pp. 111-114 (in Russian.) DOI: 10.17223/2226308X/10/44.

[3]. Devyanin P. N. System administration in MROSL DP-model. . PDM/Applied Discrete Mathematics. Supplement, 2013, no 4, pp. 22-40 (in Russian).

[4]. Devyanin P. N. Security violation necessary conditions for time information flows in MROSL DP-model. . PDM. Prilozhenie/Applied Discrete Mathematics. Supplement, no 8, 2015, pp. 81-83 (in Russian) DOI: 10.17223/2226308X/8/30.

[5]. Devyanin P. N. About results of designing hierarchical representation of mrosl DP-model. PDM. Prilozhenie/Applied Discrete Mathematics. Supplement, 2016, no 9. pp. 83-87 (in Russian) DOI: 10.17223/2226308X/9/32.

[6]. Devyanin P. N. The level of negative roles of the hierarchical representation of MROSL DP-model. PDM/Applied Discrete Mathematics, 2018, no 39, pp. 58-71 (in Russian) DOI: 10.17223/20710410/39/5.

[7]. P.N. Devyanin et al. Modeling and verification of security policies for access management in operating systems. ISP RAN, 2018, 181 p. Available at: http://www.ispras.ru/publications/security_policy_modeling_and_verification.pdf (in Russian)

[8]. Jackson D. Software Abstractions: Logic, Language, and Analysis. The MIT Press, 2012, 376 p.

[9]. Abrial J.-R. The B-book: Assigning Programs to Meanings. Cambridge University Press, 1996, 779 p.

[10]. Jean-Raymond Abrial. Modeling in Event-B. System and Software Engineering. Cambridge University Press, 2010.

[11]. P.N. Devyanin, V. V. Kulyamin, A.K. Petrenko, A.V. Khoroshilov, I.V. Shchepetkov. Comparison of specification decomposition methods in Event-B. Programming and Computer Software, vol. 42, no. 4, 2016, pp. 198-205. DOI: 10.1134/S0361768816040022.

[12]. Jones C. B. Systematic Software Development Using VDM (2nd Ed.). Prentice-Hall, Inc., 1990, 333 p.

[13]. Singh M., Sharma A. K., Saxena R. Formal Transformation of UML Diagram: Use Case, Class, Sequence Diagram with Z Notation for Representing the Static and Dynamic Perspectives of System. Proc. of the International Conference on ICT for Sustainable Developmen, 2016, pp. 25-38, DOI: 10.1007/978-981-10-0135-2_3.

[14]. Lamport L. The Temporal Logic of Actions. ACM Trans. Program. Lang. Syst, vol. 16, no. 3, 1994, pp. 872-923.

[15]. Lamport L. Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2002.

[16]. L. Lamport et al. Specifying and verifying systems with TLA+. Proc. of the 10th ACM SIGOPS European Workshop, 2002, pp. 45-48.

[17]. Lamport L. The PlusCAL Algorithm Language. Lecture Notes in Computer Science, vol. 4229, 2006, pp. 23-23. DOI: 10.1007/11888116_2.

[18]. Devyanin P. N. Security conditions for information flows by memory within the mrosl DP-model. PDM. Prilozhenie/Applied Discrete Mathematics. Supplement, issue 7, 2014, pp. 82-85 (in Russian).
[19]. Denis Cousineau et al. TLA + Proofs. Lecture Notes in Computer, vol. 7436, 2012, pp. 147-154. DOI: 10.1007/978-3-642-32759-9_14.
[20]. Kaustuv Chaudhuri et al. A TLA+ Proof System. In Proc. of the Combined KEAPPA - IWIL Workshops, 2008, pp. 17-37, URL: http://ceur-ws.org/Vol-418/paper2.pdf.
[21]. Merz S., Vanzetto H. Encoding TLA+ into Many-Sorted First-Order Logic. In Proc. of the 5th International Conference on Abstract State Machines, 2016, pp. 54-69.
[22]. Xinwen Zhang et al. Formal Model and Policy Specification of Usage Control. ACM Transactions on Information and System Security, vol. 8, no. 4, 2005, pp. 351-387. DOI: 10.1145/1108906.1108908.
[23]. Gouglidis A., Grompanopoulos C., Mavridou A. Formal Verification of Usage Control Models: A Case Study of UseCON Using TLA+. In Proc. of the 1st International Workshop on Methods and Tools for Rigorous System Design, 2018, pp. 52-64.
[24]. Stirling C. Modal and temporal logics. LFCS, Department of Computer Science, University of Edinburgh, 1991.
[25]. McMillan K. L. Eager Abstraction for Symbolic Model Checking. Lecture Notes in Computer Science, vol. 10981, 2018, pp. 191-208. DOI: 10.1007/978-3-319-96145-3_11.
[26]. Storey V. C., Song I.-Y. Big data technologies and Management: What conceptual modeling can do. Data & Knowledge Engineering, vol. 108, 2017, pp. 50-67. DOI: 10.1016/j.datak.2017.01.001.
[27]. Chris Newcombe et al. How AmazonWeb Services Uses Formal. Communications of the ACM, vol. 58, no. 4, 2015, pp. 66-73.

Спецификация модели управления доступом на языке темпоральной логики действий Лэмпорта

А. В. Козачок <a.kazachok@academ.mks.rsnet.ru>
Академия Федеральной службы охраны Российской федерации,
302015, Россия, г. Орёл, ул. Приборостроительная, д. 35

Аннотация. В статье представлено описание модели управления доступом на языке темпоральной логики действий Лэмпорта, обеспечивающей выполнение требований мандатного контроля целостности и конфиденциальности с учетом информационных потоков по памяти. Отличительной особенностью модели является учет особенностей жизненного цикла электронных документов (задания прав к метаинформации и содержимому документа отдельно, ограничение числа копий документа). Для задания модели управления доступом был выбран язык темпоральной логики действий Лэмпорта, поскольку его нотация представляется наиболее близкой к общепринятой математической, выразительные возможности и инструментальные средства позволяют описывать и верифицировать системы, заданные в виде конечных автоматов. В модели определены следующие действия: создание/удаление субъекта, чтение, запись, дозапись ("слепая" запись), создание/удаление объекта, назначение/удаление прав доступа, вложение объекта в объект, исключение вложенного объекта, утверждение объекта (документа), отправка объекта (документа) в архив, отмена действия утвержденного объекта (документа), копирование объекта (документа). Также

определены следующие инварианты: проверки типов (включает в себя проверку соответствия всех полей объектов, также проверку соответствия типу субъектов и проверку уникальности идентификаторов субъектов и объектов) и проверки безопасности (включает в себя проверку меток конфиденциальности и целостности взаимодействующих субъектов и объектов, а также корректность процедуры назначения прав доступа).

Ключевые слова: модели безопасности, компьютерные системы, верификация, моделирование, темпоральная логика, политика безопасности, управление доступом.

DOI: 10.15514/ISPRAS-2018-30(5)-9

Для цитирования: Козачок А.В. Спецификация модели управления доступом на языке темпоральной логики действий Лэмпорта. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 147-162 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-9

Список литературы

[1]. Девянин П.Н. О проблеме представления формальной модели политики безопасности операционных систем. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 7-16. DOI: 10.15514/ISPRAS-2017-29(3)-1.

[2]. Девянин П.Н. Реализация невырожденной решётки уровней целостности в рамках иерархического представления МРОСЛ ДП-модели. ПДМ. Приложение, № 10, 2017 г., стр. 111-114. DOI: 10.17223/2226308X/10/44.

[3]. Девянин П.Н. Администрирование системы в рамках мандатной сущностно-ролевой ДП-модели управления доступом и информационными потоками в ОС семейства Linux. ПДМ. Приложение, № 4, 2013 г., стр. 22-40.

[4]. Девянин П.Н. Необходимые условия нарушения безопасности информационных потоков по времени в рамках МРОСЛ ДП-модели. ПДМ. Приложение, № 8, 2015 г., стр. 81-83. DOI:10.17223/2226308X/8/30.

[5]. Девянин П.Н. О результатах формирования иерархического представления МРОСЛ ДП-модели. ПДМ. Приложение, № 9, 2016 г., стр. 83-87. DOI: 10.17223/2226308X/9/32.

[6]. Девянин П.Н. Уровень запрещающих ролей иерархического представления МРОСЛ ДП-модели. ПДМ, № 39, 2018 г., стр. 58-71. DOI: 10.17223/20710410/39/5.

[7]. П.Н. Девянин и др. Моделирование и верификация политик безопасности управления доступом в операционных системах. Институт системного программирования им. В. П. Иванникова РАН, 2018, 181 с. URL: http://www.ispras.ru /publications/2018/security_policy_modeling_and_verification.

[8]. Jackson D. Software Abstractions: Logic, Language, and Analysis. The MIT Press, 2012, 376 p.

[9]. Abrial J.-R. The B-book: Assigning Programs to Meanings. Cambridge University Press, 1996, 779 p.

[10]. Jean-Raymond Abrial. Modeling in Event-B. System and Software Engineering. Cambridge University Press, 2010.

[11]. P.N. Devyanin, V. V. Kulyamin, A.K. Petrenko, A.V. Khoroshilov, I.V. Shchepetkov. Comparison of specification decomposition methods in Event-B. Programming and Computer Software, vol. 42, no. 4, 2016, pp. 198-205. DOI: 10.1134/S0361768816040022.

[12]. Jones C. B. Systematic Software Development Using VDM (2nd Ed.). Prentice-Hall, Inc., 1990, 333 p.
[13]. Singh M., Sharma A. K., Saxena R. Formal Transformation of UML Diagram: Use Case, Class, Sequence Diagram with Z Notation for Representing the Static and Dynamic Perspectives of System. Proc. of the International Conference on ICT for Sustainable Developmen, 2016, pp. 25-38, DOI: 10.1007/978-981-10-0135-2_3.
[14]. Lamport L. The Temporal Logic of Actions. ACM Trans. Program. Lang. Syst, vol. 16, no. 3, 1994, pp. 872-923.
[15]. Lamport L. Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2002.
[16]. L. Lamport et al. Specifying and verifying systems with TLA+. Proc. of the 10th ACM SIGOPS European Workshop, 2002, pp. 45-48.
[17]. Lamport L. The PlusCAL Algorithm Language. Lecture Notes in Computer Science, vol. 4229, 2006, pp. 23-23. DOI: 10.1007/11888116_2.
[18]. Девянин П. Н. Условия безопасности информационных потоков по памяти в рамках МРОСЛ ДП-модели. ПДМ. Приложение, № 7, 2014 г., стр. 82-85.
[19]. Denis Cousineau et al. TLA + Proofs. Lecture Notes in Computer, vol. 7436, 2012, pp. 147-154. DOI: 10.1007/978-3-642-32759-9_14.
[20]. Kaustuv Chaudhuri et al. A TLA+ Proof System. In Proc. of the Combined KEAPPA - IWIL Workshops, 2008, pp. 17-37, URL: http://ceur-ws.org/Vol-418/paper2.pdf.
[21]. Merz S., Vanzetto H. Encoding TLA+ into Many-Sorted First-Order Logic. In Proc. of the 5th International Conference on Abstract State Machines, 2016, pp. 54-69.
[22]. Xinwen Zhang et al. Formal Model and Policy Specification of Usage Control. ACM Transactions on Information and System Security, vol. 8, no. 4, 2005, pp. 351-387. DOI: 10.1145/1108906.1108908.
[23]. Gouglidis A., Grompanopoulos C., Mavridou A. Formal Verification of Usage Control Models: A Case Study of UseCON Using TLA+. In Proc. of the 1st International Workshop on Methods and Tools for Rigorous System Design, 2018, pp. 52-64.
[24]. Stirling C. Modal and temporal logics. LFCS, Department of Computer Science, University of Edinburgh, 1991.
[25]. McMillan K. L. Eager Abstraction for Symbolic Model Checking. Lecture Notes in Computer Science, vol. 10981, 2018, pp. 191-208. DOI: 10.1007/978-3-319-96145-3_11.
[26]. Storey V. C., Song I.-Y. Big data technologies and Management: What conceptual modeling can do. Data & Knowledge Engineering, vol. 108, 2017, pp. 50-67. DOI: 10.1016/j.datak.2017.01.001.
[27]. Chris Newcombe et al. How AmazonWeb Services Uses Formal. Communications of the ACM, vol. 58, no. 4, 2015, pp. 66-73.

Formalizing Metamodel of Requirements Management System

[1]*D.S. Kildishev <kildishev@ispras.ru>*
[1 2 3 4]*A.V. Khoroshilov <khoroshilov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of RAS,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[3] *Moscow Institute of Physics and Technology,*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4] *Higher School of Economics.*
20, Myasnitskaya Ulitsa, Moscow 101000, Russia

Abstract. Requirements play an important role in the process of safety critical software development. To achieve reasonable quality and cost ratio a tool support for requirements management is required. The paper presents a formal definition of a metamodel that is used as a basis of Requality requirements management tool. An experience of implementation of the metamodel is discussed.

Keywords: Requirement, model, requirements management

DOI: 10.15514/ISPRAS-2018-30(5)-10

For citation: Kildishev D.S., Khoroshilov A.V. Formalizing metamodel of Requirements Management System. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 163-176. DOI: 10.15514/ISPRAS-2018-30(5)-10

1. Introduction

The development of complex systems is always a sophisticated task. The development of complex safety-critical systems, where the cost of errors is especially high, is particularly complicated. Modern best practices suggest that precise and accurate requirements management is an important element to solve that task.

Requirements managements in the context of safety-critical system development include the following aspects:

- building a catalogue of requirements;
- traceability links to sources of requirements;

- traceability links from other development artefacts like tests and code to requirements;
- configuration and version management;
- change management including change impact analysis.

The paper presents a formal definition of a metamodel that is used as a basis of Requality requirements management tool that is aimed to cover all the aspects. Implementation details of the metamodel in the tool are also discussed and future directions are considered.

2. Related works

The problem of requirements management is not a new one. This activity was known as a very important one for years. As an example we may cite a publication from 1997:

"The inability to produce complete, correct, and unambiguous software requirements is still considered the major cause of software failure today" [1].

But the requirements engineering task is still the subject of different investigations. Some of them defines a methodology [2], a model [3] or a framework [4]. Also, there are papers presenting development story of some tools, like [5].

Some papers describe both requirements model and its application in a specific tool. For example [6] designs a tool for management of requirements in form of specific models or [7] that defines some details about a feature management tool for product lines. Another paper [8] defines requirements as constraints and examine core concepts related to its implementation in a real tool.

There are many commercial requirements management tools with a little information about architecture and implementation details. There only a few open source tools are known and cited in publications like ProR [9] or ReqLine [10].

None of the papers on the tools discusses its core model in a formal way. Some approaches and models are listed in [11] but it specifies mostly methodological aspects.

3. Base model

3.1 Preliminaries

The process of software development can be made in different ways. There are some general views on requirements management tool's functions but the set of requirements for this tool in specific areas may be different.

One of the ways to deal with such problem is to develop a model for that tool. This approach can be found in [7] or [8]. The model helps to define core concepts of the tool and prove some theorems over its functions.

We need to provide some terminology before starting a model. First, we will define what the *requirement* is. In this paper, *requirement* means a limitation or definition

of some system's or component's functional. For our model *requirements* are unique objects that may have a specific description written by natural language and are placed in some tree structure defined below.

3.2 Base model

Definition 1. A *tree G* is a triple (V, E, r_0), where:

- V - a set of vertices.
- $E \subset V \times V$ - a set of edges that is an asymmetrical relation on V.
- $r_0 \in V$ - a root of the tree.
- There are no incoming edges for r_0 and there are no more than one incoming edge for the other vertices.
- All vertices are reachable from r_0.

If $(v_1, v_2) \in E$ then v_1 is denoted as a parent(v_2), while v_2 is called a child of v_1. We define relation $\text{reachable}_E(v_1, v_2)$ as a transitive and reflexive closure of the relation E.

Definition 2. *Attributed tree* AT = (G, Key, Value, attrs) consists of:

- a tree G = (V, E, r_0);
- a set of attribute keys Key;
- a set of attribute values Value;
- a functional relation attrs: V → (Key → Value) that provides each vertex with a set of attributes.

A set of all possible attributed trees is denoted as ATrees.

An attributed tree is a convenient framework to represent requirements [12] with the following semantics. If a vertex $v \in V$ represents a requirement for a target system and there are children $v_1, \ldots v_n$ of v, then the children represent a decomposition of the requirement v. In other words, if a system satisfies to requirement v then it satisfies to all requirements $v_1, \ldots v_n$ and vice versa.

Attributes of vertices contain various information about the requirements, for example a unique identifier, description of the requirements in natural language, its representation in a formal notation, version, etc.

An interesting particular case is the attributes, whose value is a vertex $v \in V$ or a set of vertices $vs \subseteq V$. It allows to define and to manage relations between different vertices. For example, such attributes can be used to represent traceability links between high level and low level requirements. Formally, this case is achieved if $V \cup \wp(V) \subseteq$ Value.

4. Declarative model

4.1 The extension of the base model

The base model of requirements catalogue is an attributed tree, where each requirement has a particular set of attributes. This model is convenient for analysis of the catalogue, e.g. for formal analysis, analysis of test coverage, traceability analysis, etc. At the same time, it is difficult to manage such model manually because there are usually many interdependencies between elements and its attributes. Here and after term vertex (element of set V) and elements of requirements catalogue are used interchangeably.

That is why we introduce a declarative model of requirements catalogue that allows us to automate the handling of such dependencies. The purpose of the declarative model is to store requirements catalogue in more compact and manageable way.

The declarative model is defined stepwise. Each step is accompanied by definition of the transformation of the declarative model to the raw basic model.

4.2 Predicates

If requirements are developed for a product line, there is a number of requirements shared between different variations of the product. A natural wish is to have a single requirements catalogue for the product line and the ability to build a specific one for a particular version of the product. That means there is a need to delete a subset of requirements from the catalogue if the subset is not applicable to the target product.

The similar situation happens when a catalogue is used to represent requirements of several revisions of a standard or to represent requirements of a standard with optional elements.

To introduce such ability we propose to choose especial key predicate $\in$ Key, whose values are boolean. If an element has attribute predicate with value false, this element and all its children are removed from the catalogue during transformation.

The first declarative model DM_1 is an attributed tree ((V, E, r_0), Key $\sqcup$ {predicate}, Value, pattrs) that is transformed to the base model ((V', E', r_0), Key, Value, attrs) according the following rules:

- $V' = \{v \in V: \forall\ v' \in V\ \text{reachable}_E(v',v)$
  $\text{predicate} \notin \text{pattrs}(v') \vee \text{pattrs}(v')(\text{predicate}) \neq \text{false}\}$;
- $E' = E \cap (V' \times V')$;
- $\forall\ v \in V'\ \text{attrs}(v) = \{(k,val) \in \text{pattrs}(v): k \neq \text{predicate}\}$

4.3 Calculated attributes

It is an often situation when attribute value depends on values of the other attributes of the same element or even on attributes of the other elements. To express such dependencies explicitly we propose the second declarative model DM_2 that is an attributed tree (G, Key, FValue, fattrs), where

- FValue = Func × Value;
- Func = ATrees × V × Key × Value → Value.

The declarative model DM_1 corresponding to the model DM_2 is an attributed tree (G, Key, Value, attrs):

∀ v ∈ V (k,val) ∈ attrs(v) iff

∃ (k, (func,fval)) ∈ fattrs(v): val = func(AT, v, k, fval)

To build such requirements model it is required to solve a set of equations defined by fattrs. A simple approach is to apply fixed point iteration, while some additional implementation details will be considered in section V. There are declarative models that define a set of equations with no solutions or with non-unique solutions. A simple but reasonable limitation that allows avoiding such models is a prohibition of cyclic dependencies between attributes.

A particular case when an attribute has a constant value val is represented in the declarative model DM_2 as a pair (prj_4, val), where prj_4 is a projection function by the fourth argument: prj_4(AT, v, k, val) = val. Please note that in DM_2 predicate is considered as a regular element of the set Key.

4.4 Attribute scope

Another often situation happens when an attribute is applicable to the whole subtree and it has the same value for all elements. Or a similar case is when an attribute is applicable to all children of the particular element.

To handle such situations we propose the third declarative model DM_3 that is an attributed tree (G, Key, SValue, sattrs), where SValue = FValue × Scope, Scope = {S_L, S_{DC}. S_S} with an element having the following semantics:

- S_L – an attribute is available only in the element where it is defined.
- S_{DC} – an attribute is available in the element where it is defined and in all its direct children.
- S_S – an attribute is available in the element where it is defined and in all its successors.

An example of attribute scope can be seen on Fig. 1. White rectangles are Vs. Arrows mean child-parent relation. Attribute with some scope is defined in r_0. Grey rectangles represent different possible scopes of A and the subtrees where it will be accessible.

\A transformation of declarative model DM_3 to the model DM_2 is straightforward: DM_2 is an attributed tree (G, Key, FValue, fattrs), where fattrs(v) = {k → fval} such that

- (1) {k → (fval, anyscope)} ∈ sattrs(v)
- (2) {k → (fval, S_{DC})} ∈ sattrs(parent(v)) if rule (1) is not applicable,
- (3) {k → (fval, S_S)} ∈ sattrs(v') if rules (1) and (2) are not applicable ∧ $reachable_E$(v',v) ∧

$\forall$ v" $\in$ V $\forall$ val $\in$ Value (reachable$_E$(v",v) $\wedge$ reachable$_E$(v',v")) $\Rightarrow$ {k $\rightarrow$ (val, S_S)} $\notin$ sattrs(v").

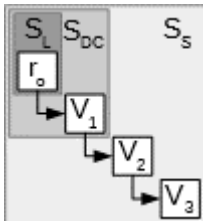

Fig. 1. Attribute scopes

It is interesting to note that nonconstant scoped attributes can get different values in different elements because its function can depend on the vertex as a third argument.

4.5 Reuse of subtrees

The next item to consider is a situation when there are several subtrees of requirements that are very similar each other up to some limited number of details. In this case, it would be ideal to have a single copy of the subtree and the ability to clone it with some modifications. This approach is usually called reuse [13].

The fourth declarative model DM_4 is an attributed tree ((V, E, r_0), Key $\sqcup$ {cp}, SValue, cpattrs) with especial key cp that satisfies the following constraints:

- $\forall$ v $\in$ V$\forall$ value $\in$ Value$\forall$ s $\in$ Scope cp $\in$ cpattrs(v) $\wedge$ cpattrs(v)(cp) = ((prj$_4$,val), s) $\Rightarrow$ val$\in$ V $\wedge$ $\forall$ v' $\in$ V (v, v') $\notin$ E;
- E $\cup$ {(v, cp(v))| v $\in$ CC(DM_4)} does not contain loops, where CC(DM_4) = {v $\in$ V| cp $\in$ cpattrs(v)} and cp(v) - val$\in$ V from the constraint above.

The transformation of the model DM_4 to the model DM_3 = ((V', E', r_0), Key, SValue, sattrs) is performed by the following algorithm:

1. curDM$_4$:= DM$_4$
2. If CC(curDM$_4$) is empty, take DM$_3$ = curDM$_4$ with removing cp from the Key set and finish.
3. Let curDM$_4$ is ((V, E, r_0), Key $\sqcup$ {cp}, SValue, cpattrs).
4. Choose any v_0 $\in$ CC(curDM$_4$) such that $\nexists$ v $\in$ CC(curDM$_4$) reachable$_E$(cp(v_0),v). Existence of such element follows from lack of loops in E $\cup$ {(v, cp(v))| v $\in$ CC(DM_4)}.
5. Assume without loss of generality $\forall$ v$\in$ V (v_0,v)$\notin$ V.
6. Build newDM$_4$ = ((V', E', r_0), Key $\sqcup$ {cp}, SValue, cpattrs'), where
   - V' = V $\cup$ {(v_0,v') | v'$\in$ V $\wedge$ reachable$_E$(cp(v_0),v') }

- $E' = E \cup \{ (v_0,(v_0,cp(v_0))) \} \cup \{ ((v_0,v'),(v_0,v'')) \mid (v',v'') \in E \wedge reachable_E(cp(v_0),v') \}$
- $\forall\ v \in V\backslash\{v_0\}\ cpattrs'(v) = cpattrs(v)$
- $cpattrs'(v_0) = \{(k,val) \in cpattrs(v_0): k \neq cp\}$
- $cpattrs'((v_0,v')) = cpattrs(v')$

Please note that $newDM_4$ satisfies both constraints of the fourth declarative model.

7. $curDM_4 := newDM_4$ and goto step 2.

Lemma 1. The algorithm terminates for any DM4 satisfying the constraints.

The proof is based on the fact that the cardinality of $CC(curDM_4)$ is decreased every iteration because of the choice of the v_0 at step 4 that guarantees that elements with attribute cp are not cloned, while one such element loses that attribute.

Lemma 2. The result of transformation does not depend on the order of the selection of elements at step 4.

The idea of the proof is that transformations that can be chosen in non-deterministic order make modifications in non-intersecting subtrees.

Interesting to note that combination of reuse and predicate transformation can be used to define a generic subtree that is instantiated several times with different arguments using reuse transformation and the original generic subtree is eliminated with predicate transformation. Also, predicate transformation can be useful to eliminate unneeded elements from the cloned subtrees.

5. Implementation details

5.1 Identification

One of the important aspects of requirements management is requirements identification. One of the common approaches it to assign a unique identifier to each object, for example, some number or string.

In addition to that it is possible to provide each element with a qualified identifier QID defined recursively on top of identifiers ID that are unique within children of the same parent: r_0 has QID = '/ID', child v has QID = 'QID(parent)/ID'.

Let us take some example of requirements for some system. If we use QID we can have a human-readable path for each requirement. For example, we may have an element with QID = "Functional requirements/Ports/req001". As seen from the path it has a parent "Functional requirements/Ports/" and its ID is req001.

5.2 Calculated attributes

There are two objects related to attributes in the implementation. The first one, *attribute definition* A_DEF represents a pair (func,fval) from the formal declarative mode, where func is of type ATrees × V × Key × Value → Value. The second one,

attribute A, represents a value of the attribute in the base model. A_DEF is used to calculate an actual value A when it is required.

There are several kinds of functions supported in attribute definitions.

The first kind is the *constant* functions prj_4 that always returns fval value stored in attribute definitions.

The second kind is *template functions* that stores in fval value a string with parameters encoded in curly brackets, e.g. "Hello, {K}". The value of the parameters to be used for substitution is taken from attribute with the encoded name, 'K' in the example above, of the same element.

The third kind is *formula value generator* that stores in fval value a string with an expression in a subset of JavaScript language that has access to attributes of the same element.

The fourth kind is *virtual attributes* that are implemented in Java. They have no stored fval value at all, but they have access to the whole context of the element including the complete attributed tree.

For example, Label attribute can take value of user-defined Name attribute if there is one or return system-defined identifier otherwise. Another example could be QID that calculates qualified identifier of the element as a concatenation via '/' of parent's QID with a Name of the target element.

An important additional information that the tool is able extract from attribute definition is a set of attribute keys which values are required to calculate the actual value for the given attribute by the corresponding function.

5.3 Attributes life-cycle

For each attribute stored data includes function kind and fval. The pair (func-kind,fval) is denoted A_ST. System-defined virtual attributes have no stored data, they are added to elements on the fly.

Let us describe a common process of attributes loading for some requirement.

1. Set of A_ST is loaded from storage to A_DEFS.
2. Set of scoped attributes that are applicable to the target one is taken from its parent and is added to A_DEFS.
3. The A_DEFS set is handled by Attribute_Calculation procedure described below.

If attributes are changed by the user using GUI session, the tool has the same A_DEFS set that contains a subset of changed attribute definitions. Then the tool applies the same Attribute_Calculation procedure as follows.

1. A_DEFS set is extended with attributes of the target element that depends on any attribute already belonging to A_DEFS.
2. The order of evaluation of attributes from A_DEFS is calculated. The order can be defined as ORDER = $(K_1..K_n)$ where K_i is the key of the attribute.

-$\forall$ Ki,Kj$\in$ ORDER if K_j depends on K_i then i<j.

The algorithm is described in the next section.

3. For each A_DEF in the A_DEFS value of A is calculated and placed to AS.

After this procedure AS contains an actual state of attributes after provided changes.

5.4 Order extraction algorithm

As an input of order extraction, we have KEYS = (K_1.. K_n) that is set of attributes name in some random order and DEPS = ($K_i \rightarrow (K_{j1}..K_{jm})$) - a map of attributes dependencies. The algorithm is as follows:

1. ORDER is set to empty collection.
2. OSET is the set of handling nodes.
3. Extract revert dependencies DEPS_R. DEPS_K=($K_j \rightarrow (K_{i1}.. K_{il})$). If K_i depends on K_j then DEPS_K contains $K_i \rightarrow K_j$ record.
4. Place KEYS to OSET.
5. Set flag MOD is to False.
6. In OSET look for candidate KK with DEPS_K[KK] = KSET that complies one of following rules:
   - KSET is empty.

OR

   - $!\exists K_i \in$ KSET: $Ki \in$ OSET.
7. If K_K was found then:
   1. MOD set to True.
   2. K_K removed from OSET.
   3. K_K added to ORDER.
8. If MOD = True & |SET|!= 0 then go to step 4.

At the end of execution, the ORDER will contain the order in which A's values calculation.

5.5 Attribute change management

The introduction of scope and calculated attributes requires the management of attributes changes to keep all dependent attributes up-to-date.

There are two possible strategies to deal with attribute updates. The first approach is to commit all changes at runtime. The second one is to collect changes in AS and then apply them all by request. Immediate commit is tending to be simpler but more computing - intensive. Late updates require fewer calculations but need more memory. For our tool, we use the second approach because we have large catalogues with a possibility of complex relationships between its elements.

Late changes can be defined in form of new object — changes set CS=($K \rightarrow OP$, $K \rightarrow A_{OLD}$, $K \rightarrow A_{NEW}$) where K is the key of attribute, $OP \in$ (remove, create, modi-

fy) is the operation over attribute, A_{OLD} is the value of attribute in AS before operation, A_{NEW} is the new attribute value after operation.

For attribute changes change set needs to store A_DEFS, so minimal CS = (K, K→ A_{OLD}, K→ A_{NEW}). To use these changes set we need to extend the model of attributes set of A.

When all attribute modifications are collected we need to apply all that changes to calculate actual values of attributes. It is implemented in the same way as it was described in section V.C.

One more problem with attribute changes is that some of the changes need to be propagated from one requirement to another. To deal with this problem we define a concept of *change propagators*. If A_DEF (virtual attributes only for now) depends on attributes from the external element it registers a function-change propagator that is called when some change set is applied to attributes of that element. The change propagator evaluates if the changes impact the target attribute and initiate its recalculation if it is required.

5.6 Lazy loading

When we speak about a model of requirements in some common application like avionics we need to take into account the number of distinct requirement. Sometimes the number of artefacts for such models tends to be in the thousands or tens thousands. In that case, direct management of requirements may require a lot of resources.

To solve this problem we use the lazy loading principle. That means that AT will contain only those Vs that are requested during the usage of the model. In most cases that means that in G we have a subtree $G_L \in G$ that contains r_0 and some subtrees that are used during the current working session.

But laziness of model leads to some difficulties. First of all, we need to overlook AT instead of AT_L if we need to assure that V with given ID exists. This problem can be solved by caching id-related information in CacheStore that is always available.

5.7 Attribute types

In practice, the value of an attribute may have one more property – a *type*. One possible set of types includes Integer, Boolean, String, Float. Also, we may define types for Collection and Enumeration. In most cases, the value still is the simple constant.

But some attribute types cannot be defined as a single value and need to store and manage some additional data. For example, *Collection* type may use specific object LIST = (T_V, V_1..V_n) where T_V is the type of collection's value and (V_1..V_n) are the values stored in the collection.

One more specific type is *Enumeration*. First, enumeration requires definitions of its values. It can be made by means of ENUM_DEF = (V_T, V_1...V_n) that is similar to LIST one. But to define an attribute with one selected enumeration value we need to define one more object ENUM = (K_B, V_S) where K_B is the key of A with

ENUM_DEF and V_S is the selected value. But in a case we introduce an ENUM, we need to ensure that for every ENUM we will have an A_D where T = ENUM_DEF and V_D will contain V_S.

5.8 References

One more problem is the implementation of relations between elements of the catalogue. Some tools manage them as the set of specific objects placed in the distinct set.

In our model relations are presented in form of specific attribute type REFERENCE. For this type we introduce value object REF_VALUE (REF, V, ERR) where REF is a string that can be resolved to V, usually containing some kind of identifier, V is the corresponding element if there is any matched by identifier, ERR is a string with an error message if REF cannot be resolved or contains incorrect value.

In this case, REF_VALUE initially contains only REF field. If someone requires the result of REF_VALUE resolution then the tool tries to resolve the REF and then fills V or ERR.

References are also required some additional handling to support its consistency. In a case REF or target V is changed we may need to track its changes and update related REF_VALUE.

One more specific problem is reverted links. If we have a relation $V_1 \rightarrow V_2$ we may need to know for V_2 that it has a relation to V_1. This kind of relations is called "reverse references".

If links are stored in AT then we may use one more function $(V_2, LN) \rightarrow V_1$ to store reverse relations. If we define a new type of attributes or the specific state of REF_VALUE then we face a problem of keeping it up-to-date.

In our model, we store reverts links in the cache in form of $(V_2, LN) \rightarrow (V_1 \ldots V_n)$ function. That allows us to easily get revert links on V_2 if the state of cache is valid.

In a case of completely loaded AT the problem is not so difficult to solve because we always have the actual state of every V. But we cannot guaranty the V's state in case of a partially loaded AT that happens in case of lazy loading.

If we have some loaded $AT_L \subseteq AT$, relation $(V_1, LN) \rightarrow V_2$, $V_1 \notin AT \wedge V_2 \in AT$ then if we need to get revert links on V_2 we may need to load the whole AT to be sure that all possible V_1 were found.

In our case, this problem is solved by storing reverse links in the cache. But in this case, we still have one necessary problem. Let us introduce some link $L(V_1, V_2, LN)$. If we already resolve this link then the record in cache tends to be present. But what if we introduce V_2 in the model when V_1 is loaded and the link is resolved was not found? The situation takes place when V_2 is loaded by the lazy method, created or modified.

In the worst case, we need to track changes of the whole AT for all links. A better solution is to manage some kind of scope for which link tends to be resolved. That is not implemented yet, but it is in our plans.

Relations can be used for some specific activities. One of them is changes management. Changes management is performed when some V_1 with links ($L_1..L_n$) is changed. In this case, some operations will be performed on V's obtained from $L_1..L_n$. The nature of such operation can be different. For some tools, those Vs will be marked in a model with the specific flag. In other cases, the models can define additional actions depending on the kind of change.

Conclusion

We presented a formal metamodel that is used as a basis for building Requality requirements management tool. We covered different difficulties related to its implementation. But the experience demonstrates that the model allows handling quite big requirements catalogue with many relations between its elements.

The future work includes analysis and implementation of new kinds of functions for calculated values and development of user-friendly patterns for solving common user tasks on top of the semantics defined in the paper. Another direction is research of possible compositions of the formal model provided by the tool and formal models used to represent particular requirements.

Acknowledgment

This study was supported by RFBR grant #17-07-00734.

References

[1]. R. Thayer. M. Dorfman. Software Engineering. IEEE Computer Society press, 1997., 552 p.

[2]. M. Palumbo. Requirements Management for Safety Critical Systems. Available: http://www.railwaysignalling.eu/wp-content/uploads/2015/06/Req_mgt_safety_critical_system_Mpalumbo.pdf. Accessed: 3-Apr-2018

[3]. P. Roques. Modeling Requirements with SysML. Requirements Engineering Magazine, issue 2015-02, 2015.

[4]. Open Group Standard. Dependability through Assuredness (O-DA) Framework. The Open Group Releases, 2013.

[5]. A. Nordin, A. Ikhwan Omar, M. Usamah Megat Mohamed Amin, N. Salleh. .Development of scenario management and requirements tool (SMaRT): towards supporting scenario-based requirements engineering methodology. International Journal of Engineering & Technology, Vol 7, No 2.14, Special Issue 14, 2018, pp 62-65.

[6]. D. Lozhkina, S. Staroletov. An online tool for requirements engineering, modeling and verification of distributed software based on the MDD approach. In Preliminary Proceedings of the 11th Spring/Summer Young Researchers'Colloquium on Software Engineering, 2017, pp. 23-28.

[7]. T. von der Maßen, H. Lichter. RequiLine: A Requirements Engineering Tool for Software Product Lines, Software Product-Family Engineering, 2003, Heidelberg, pp. 168-180.

[8]. N. W. Mogk. A Requirements Management System based on an Optimization Model of the Design Process. In Proc. of the Conference on Systems Engineering Research (CSER 2014), 2014, pp 21-22
[9]. ProR Requirement Engineering Platform. [Online]. http://www.eclipse.org/rmf/pror/. Accessed: 2-Apr-2018.
[10]. ReqLine Download (ReqLine.exe). [Online]. Available: http://downloads.informer.com/reqline/. Accessed: 3-Apr-2018.
[11]. S. Hallerstede, M. Jastram, L. Ladenberger. A method and tool for tracing requirements into specifications. Science of Computer Programming, vol. 82, 2014, pp. 2–21.
[12]. Alexey Khoroshilov. On formalization of operating systems behaviour verification. In Proceedings of 11th International Conference on Computer Science and Information Technologies (CSIT-2017), 2017, pp. 168-172. DOI:10.1109/CSITechnol.2017.8312164
[13]. W. Frakes, C. Terry. Software Reuse: Metrics and Models. ACM Computing Surveys Vol. 28, No. 2, 1996.

Формализация метамодели системы управления требованиями

[1] *Д.С.Кильдишев <kildishev@ispras.ru>*
[1,2,3,4] *А.В.Хорошилов <khoroshilov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[3] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[4] *Высшая школа экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. В рамках данной статьи рассматривается **метамодель**, лежащая в основе системы управления требованиями Requality. Базовая модель представляет собой дерево, каждой вершине которого сопоставлен набор именованных и типизированных свойств. Базовая модель проста и удобна для представления семантики набора требований, но оказывается не особо пригодной для формирования и сопровождения сколько-нибудь сложных каталогов требований. Поэтому авторами вводится набор декларативных моделей, позволяющих описывать каталог требований более компактным образом. При этом семантика декларативных моделей задаётся при помощи определения трансляции в базовую модель. Эти возможности обеспечивают гибкий инструментарий для компактного описания типовых наборов требований. Также в статье рассматриваются особенности реализации представленной метамодели в системе управления требованиями Requality. В заключении предлагается исследовать комбинацию представ-

ленной модели каталога требований с формальными моделями, позволяющими описывать семантику каждого требования в отдельности.

Ключевые слова: требование; модель; управление требованиями

DOI: 10.15514/ISPRAS-2018-30(5)-10

Для цитирования: Кильдишев Д.С., Хорошилов А.В. Формализация метамодели системы управления требованиями. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 163-176 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-10

Список литературы

[1]. R. Thayer. M. Dorfman. Software Engineering. IEEE Computer Society press, 1997., 552 p.

[2]. M. Palumbo. Requirements Management for Safety Critical Systems. Available: http://www.railwaysignalling.eu/wp-content/uploads/2015/06/Req_mgt_safety_critical_system_Mpalumbo.pdf. Accessed: 3-Apr-2018

[3]. P. Roques. Modeling Requirements with SysML. Requirements Engineering Magazine, issue 2015-02, 2015.

[4]. Open Group Standard. Dependability through Assuredness (O-DA) Framework. The Open Group Releases, 2013.

[5]. A. Nordin, A. Ikhwan Omar, M. Usamah Megat Mohamed Amin, N. Salleh. .Development of scenario management and requirements tool (SMaRT): towards supporting scenario-based requirements engineering methodology. International Journal of Engineering & Technology, Vol 7, No 2.14, Special Issue 14, 2018, pp 62-65.

[6]. D. Lozhkina, S. Staroletov. An online tool for requirements engineering, modeling and verification of distributed software based on the MDD approach. In Preliminary Proceedings of the 11th Spring/Summer Young Researchers'Colloquium on Software Engineering, 2017, pp. 23-28.

[7]. T. von der Maßen, H. Lichter. RequiLine: A Requirements Engineering Tool for Software Product Lines, Software Product-Family Engineering, 2003, Heidelberg, pp. 168-180.

[8]. N. W. Mogk. A Requirements Management System based on an Optimization Model of the Design Process. In Proc. of the Conference on Systems Engineering Research (CSER 2014), 2014, pp 21-22

[9]. ProR Requirement Engineering Platform. [Online]. http://www.eclipse.org/rmf/pror/. Accessed: 2-Apr-2018.

[10]. ReqLine Download (ReqLine.exe). [Online]. Available: http://downloads.informer.com/reqline/. Accessed: 3-Apr-2018.

[11]. S. Hallerstede, M. Jastram, L. Ladenberger. A method and tool for tracing requirements into specifications. Science of Computer Programming, vol. 82, 2014, pp. 2–21.

[12]. Alexey Khoroshilov. On formalization of operating systems behaviour verification. In Proceedings of 11th International Conference on Computer Science and Information Technologies (CSIT-2017), 2017, pp. 168-172. DOI: 10.1109/CSITechnol.2017.8312164.

[13]. W. Frakes, C. Terry. Software Reuse: Metrics and Models. ACM Computing Surveys Vol. 28, No. 2, 1996.

Сравнительный анализ нейронных сетей в задаче классификации побочных эффектов на уровне сущностей в англоязычных текстах

И.С. Алимова <alimovailseyar@gmail.com>
Е.В. Тутубалина <tutubalinaev@gmail.com >
Казанский (Приволжский) федеральный университет,
420008, Казань, ул. Кремлёвская, 18

Аннотация. В данной работе представлено экспериментальное исследование эффективности ряда моделей нейронных сетей для задачи классификации побочных эффектов на уровне сущностей. Задача анализа тональности на уровне аспектных терминов, в которых необходимо определить мнение по отношению к конкретному аспекту, активно исследуется в течении последнего десятилетия. Для решения данной задачи в прошедшие годы было предложено несколько архитектур нейронных сетей. Несмотря на то, что модели, основанные на этих архитектурах, имеют много общего, есть некоторые компоненты, которые отличают их друг от друга. В данной статье была исследована применимость разработанных для аспектно ориентированного анализа тональности нейросетевых моделей для классификации побочных эффектов. Для оценки эффективности данных методов были проведены обширные эксперименты на различных англоязычных текстах биомедицинской тематики, включающих в себя записи клинических карточек, научную литературу и данные из социальных сетей. Также мы сравнили предлагаемую модель с одной из наилучших на данный момент моделей, основанной на методе опорных векторов и большом наборе признаков.

Ключевые слова: побочный эффект; обработка естественного языка; анализ социальных медиа; машинное обучение; глубокое обучение; нейронные сети

DOI: 10.15514/ISPRAS-2018-30(5)-11

Для цитирования: Алимова И.С., Тутубалина Е.В. Сравнительный анализ нейронных сетей в задаче классификации побочных эффектов на уровне сущностей в англоязычных текстах. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 177-196. DOI: 10.15514/ISPRAS-2018-30(5)-11

1. Введение

В настоящее время в связи с бурным развитием сети интернет и электронных коллекций научных публикаций имеются обилие неструктурированной

информации, представленной текстами на естественном языке. В число активно развивающихся направлений обработки текстовой информации входят задачи медицинской науки, в частности, задачи фармакологии и персонализированной медицины. Всё более востребованной становится задача автоматической обработки текстов медицинской направленности с целью извлечения структурированных данных, которые затем используются при решении различного рода проблем: поиска информации о побочных реакциях лекарственных препаратов, использовании лекарств с нарушением предписаний инструкции, определении действия лекарственных препаратов по отношению к системам организма, излечения новых отношений между лекарствами и симптомами для построения гипотез о перепрофилировании препарата.

Для выявления новых побочных эффектов, не указанных в инструкции по применению препарата, все большую популярность приобретает подход с применением текстов медицинской тематики: электронных карточек пациентов, научной литературе, записей пациентов в социальных сетях и медицинских форумах. Обработка такого объема информации невозможна вручную, поэтому активно применяются методы автоматической обработки естественного языка [1-5].

Классификацию побочных эффектов можно рассматривать в двух направлениях: (i) на уровне сообщения и (ii) на уровне сущности. В первом случае необходимо определить наличие упоминания побочного эффекта во фрагменте текста, например, предложении или тексте твита. Данный тип классификации необходим для очистки коллекции текста от нерелевантных документов. Во втором случае классификация применяется к результатам работы алгоритмов извлечения именованных сущностей. В данной работе мы сосредоточились на второй задаче.

Одна из разновидностей задач классификации относительно сущностей - это аспектно-ориентированный анализ тональности. В аспектно-ориентированном анализе тональности определяется отношение пользователя не только к объекту в целом, но и к отдельным его частям или аспектам. Существующие работы показали успешность применения ряда архитектур нейронных сетей, основанных на сетях с короткой долгосрочной памятью (англ. long short-term memory; LSTM). В данной статье разработанные методы были адаптированы и применены для задачи классификации побочных эффектов.

Исследования были начаты с простых моделей, использующих только LSTM, далее архитектуры расширялись механизмами внимания и дополнительной памятью. В качестве моделей были взяты следующие архитектуры нейронных сетей:

i. сеть с короткой долгосрочной памятью (англ. long short-term memory; LSTM) - базовая модель, которая использует все предложение, закодированное векторным представлением слов, в качестве входа;

ii. модель с заданной целью (англ. Target-Dependent LSTM; TD-LSTM) [6] которая использует два слоя LSTM для моделирования правого и левого контекста относительно сущности;

iii. сеть с механизмом интерактивного внимания (Interactive Attention Network; IAN) [7], которая состоит из двух слоев LSTM для представления предложения и целевой сущности и слоев с перекрестным вниманием, объединенные выходы которых передаются слою с логистической функцией для принятия решения о классификации;

iv. сеть с глубокой памятью (Deep Memory Network; MemNet) [8], которая применяет несколько раз механизм внимания к входному слою векторного представления слов, выход последнего из которых передается в слой с логистической функцией для предсказания класса;

v. сеть с рекуррентным механизмом внимания к памяти (Recurrent Attention Memory; RAM) [9] расширяет модель MemNet дополнительными слоями LSTM и многократным применением механизма внимания к выходам этих слоев.

Описанные модели применялись в задаче классификации мнений для отзывов пользователей о ресторанах и ноутбуках, однако работ по применению моделей к классификации побочных эффектов на уровне сущностей из различных источников текста фармаконадзора найдено не было. В рамках данного исследования были проведены обширные эксперименты на пяти базовых наборах данных, которые состоят из текстов аннотаций биомедицинских статей, электронных карточек пациентов и текстов из социальных сетей. Проведено сравнение эффективности описанных нейронных сетей и метода на основе опорных векторов с точки зрения стандартных метрик качества классификации.

2. Обзор существующих подходов

В исследованиях применяются различные подходы для выявления побочных реакций в текстах. Наиболее широко используемый метод - это подход, основанный на словарях [10-15]. Словари состоят из списков побочных реакций, извлеченных из инструкций по применению лекарств, записей о клинических испытаниях, отзывах пользователей в социальных сетях. Первые работы были ограничены в количестве исследуемых лекарств и целевых побочных эффектов из-за ограничений терминов в словарях. Для преодоления этого ограничения стали использоваться методы на основе правил [16-17]. Основная идея этих методов заключается в том, чтобы выделить наиболее распространенные конструкции предложений, которые могут свидетельствовать об описании побочных реакций. Однако разработка правил является длительным и трудоемким процессом, для этого требуется наличие специалиста в данной предметной области, при этом данный подход не масштабируем для новых коллекций документов.

Большинство работ описывают исследования с использованием методов машинного обучения. Например, в работах [18-23] используется метод опорных векторов (SVM), в статьях [16, 24] применяется метод условных случайных полей (CRF), а в работе [25] метод случайного леса (Random Forest). В качестве признаков, подаваемых на вход алгоритмам машинного обучения, используются: n-граммы, части речи, принадлежность к семантическим типам из унифицированного языка медицинских систем (UMLS), количество слов с отрицанием, принадлежность рассматриваемого термина к словарям побочных реакций, наличие в тексте названия лекарства, вектора word2vec, вектора кластеризации.

В 2016-м и 2017-м годах проводились соревнования по поиску побочных эффектов в сообщениях из Твиттера [26-27]. В рамках соревнования присутствовали задачи классификации на уровне всего твита и на уровне сущности. Победители первого соревнования использовали комбинацию из девяти классификаторов, основанных на модели случайного леса [27] со следующим набором признаков: 1, 2, 3 - граммы, появление вместе лекарства и побочного эффекта, наличие отрицания и оценка тональности. В качестве набора данных для каждого классификатора на вход подавались все положительные примеры и такое же количество случайным образом выбранных отрицательных примеров, что позволило участникам решить проблему несбалансированности классов. Описанная система получила 41.95% F-меры. В соревновании 2017-го года в задаче классификации на уровне твитов первое место заняла система, использовавшая метод опорных векторов в качестве модели [28]. Однако, в отличие от предыдущего года, набор признаков был более обширным, модель получила 43.5% F-меры и таким образом улучшила результаты предыдущего соревнования на 1.55%. В задаче классификации на уровне сущностей лучшие результаты показала система, использовавшая ансамбль сверточных нейронных сетей [29]. Система достигла 69.3% F-меры.

В 2016-м году появляются первые работы по классификации текстов на наличие побочных эффектов, основанные на нейронных сетях. В работе [30] применялись сверточная рекуррентная нейронная сеть и сверточная сеть с вниманием. Эксперименты проводились на двух наборах данных: твитов из соревнования 2016-го года, описанном в данном разделе выше [26] и отчетов системы MEDLINE [31]. Сверточная рекуррентная нейронная сеть показала 51% F-меры на корпусе твиттеров и 87% F-меры на корпусе MEDLINE, модель с вниманием показала 49% и 83% F-меры соответственно. Таким образом, был получен прирост на 7.5% в сравнении с результатами соревнования.

Методы по анализу тональности активно применяются в предметной области медицины и в текстах других тематик [32-35]. В области аспектно ориентированного анализа тональности активно применяются нейронные сети [36]. Танг и др. представили архитектуру нейронной сети TD LSTM (Target-Dependent LSTM; TD_LSTM) [6] и сеть с памятью MemNet (Deep Memory

Network; MemNet) для классификации на уровне аспекта [8]. Предложенные модели демонстрируют сравнимые с существующими методами результаты. Чен и др. использовали рекуррентную сеть с вниманием (Recurrent Attention Memory; RAM) [9]. Модель применяет механизм внимания несколько раз для охвата тональных признаков, находящихся на большом друг от друга расстоянии. RAM превзошла результаты описанных ранее моделей на четырех корпусах из разных предметных областей. Ма и др. предложили сеть с интерактивным вниманием (Interactive Attention Network; IAN), которая генерирует отдельные представления для контекста и аспекта и применяет для них перекрестное внимание [7]. Модель показала высокую эффективность по сравнению с различными модификациями нейронной сети с длинной короткой памятью (Long Short Term Memory; LSTM).

На основе анализа предметной области можно сделать вывод, что сравнительно мало работ посвящено применению нейронных сетей в задаче классификации побочных эффектов. Большинство работ используют методы машинного обучения, которые ограничены линейностью модели и необходимостью поиска оптимальных признаков вручную [2,12,18,21,25,27,37,38]. Кроме того большинство методов извлекали признаки непосредственно из классифицируемой сущности, уделяя малое внимание контексту или используя маленький контекст размером в 4-5 слов слева и справа относительно сущности [21,25,39,40]. Стоит также отметить, что в большинстве работ проводились исследования на одном корпусе данных.

3. Корпуса

Эксперименты по оценке эффективности методов классификации проводились на четырех существующих англоязычных корпусах: CADEC, Твиттер, MADE, Twimed. Общая статистика для всех корпусов представлена в табл. 1. В таблице класс ‘ADR’ обозначает класс с побочным эффектом, соответственно, класс ‘non-ADR’ обозначает его отсутствие. Как видно из статистики, корпуса CADEC и MADE содержат большее кол-во аннотаций, чем остальные корпуса.

3.1 CADEC

Корпус CADEC состоит из размеченных отзывов пользователей о лекарственных препаратов с форума askapatient.com [41]. В корпусе размечены 5 видов аннотаций: лекарство (drug), побочный эффект (adverse), заболевание (disease), симптом (symptom) и другие медицинские термины, не вошедшие в описанные категории (finding). Аннотацией лекарство отмечены все названия лекарственных препаратов в тексте. Все побочные эффекты, связанные с лекарством, отмечены аннотацией побочный эффект. Аннотацией заболевание обозначены показания к применению. Симптом обозначает сопутствующие признаки болезни. Аннотации заболевание и

симптом были сгруппированы вместе с аннотацией, обозначающей другие медицинские термины в одну группу.

3.2 Twitter

Корпус Twitter содержит твиты пользователей на тему здоровья [42]. В каждом твите отмечены побочные эффекты или сущности, обозначающие заболевание. Политика Твиттера не позволяет хранить и распространять твиты в открытом доступе. Создатели корпуса предоставляют только идентификатор пользователя и твита по которым можно загрузить исходный текст. В связи с этим часть твитов не удалось загрузить. Во время предобработки текста были удалены все ссылки, упоминания пользователей и ретвиты.

3.3 MADE

Корпус MADE состоит из обезличенных записей электронных карточек пациентов, больных раком. Корпус был создан для соревнования по обработке естественного языка, в задачи которого входило извлечение медицинских терминов, побочных эффектов и отношений между ними [43]. Аннотации, связанные с заболеваниями ‘SSLIF’ и ‘Indication’, были объединены в класс ‘non-ADR’.

3.4 Twimed

Корпус Twimed состоит из двух частей: твитов пользователей и текстов статей с ресурса PubMed [44]. Корпус содержит аннотации: болезнь, симптом и лекарство. Если отношение между лекарством и болезнью было размечено как негативное, то болезнь отмечалась как побочный эффект.

Табл. 1. Суммарная статистика по корпусам
Tab. 1. Summary statistics of corpora

Корпус	Источник	Кол-во документов	Кол-во ADR	Кол-во non-ADR	Максимальная длина предложения	Средняя длина предложения
CADEC [41]	Отзывы на форуме	1231	5770	550	236	28
MADE [43]	Электронные карточки пациентов	876	1506	37077	173	21
TwiMed-Pubmed [44]	Аннотации статей	1000	264	983	150	39
TwiMed-Twitter [44]	Твиттер	637	329	308	42	27
Twitter [42]	Твиттер	645	569	76	37	22

4 Архитектуры нейронных сетей

В данном разделе описаны архитектуры сравниваемых нейронных сетей.

4.1 LSTM

Классическая нейронная сеть, являющаяся разновидностью рекуррентных нейронных сетей, была представлена в [45]. Сеть состоит из трех слоев: входного, слоя с короткой долгосрочной памятью (LSTM) и выходного. В первом слое сети (Embedding) происходит кодирование входного текста в векторное представление и передаются в слой LSTM. Данный слой считывает пословно входное предложение и сохраняет скрытые состояния с помощью. После прочтения всего предложения скрытые состояния передаются в качестве признака в выходной классифицирующий слой с функцией softmax.

4.2 TD_LSTM

Данная модель была предложена в работе [6] и является расширением предыдущей модели. Модель состоит из двух частей, каждая из которых обрабатывает левый и правый контексты соответственно. Аналогично с предыдущей моделью входные тексты попадают в слой векторного представления слов, выходы которого передаются в LSTM слой. Вектора скрытых состояний LSTM слоев для левого и правого контекстов конкатенируются в один вектор. К полученному вектору, так же как и в предыдущей модели, применяется слой с функцией softmax и вычисляется класс с наибольшей вероятностью. Схема архитектуры данной сети представлена на рис. 1.

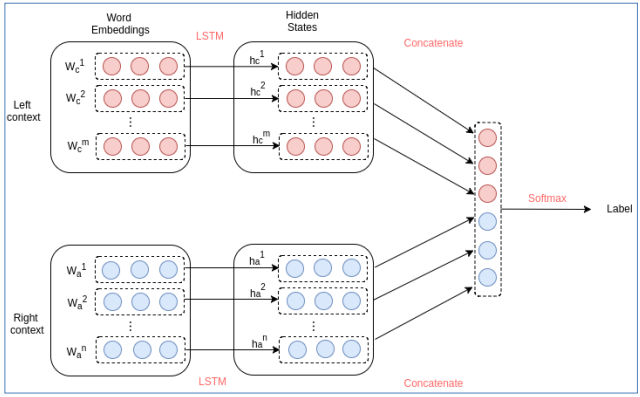

Рис. 1. Архитектура модели TD_LSTM
Fig. 1. The overall architecture of TD LSTM

4.3 IAN

Модель с механизмом интерактивного внимания была представлена в работе [7]. Сеть состоит из двух частей, каждая из которых строит представление контекста и классифицируемой сущности с помощью векторного представления слов и LSTM слоя. Полученные вектора усредняются и используются для вычисления вектора внимания. В первом слое внимания используется вектор контекста и усредненный вектор сущности, во втором - вектор сущности и усредненный вектор контекста. Полученные на выходе вектора конкатенируются и передаются слою с функцией активации softmax для классификации. Схема архитектуры сети представлена на рис.2.

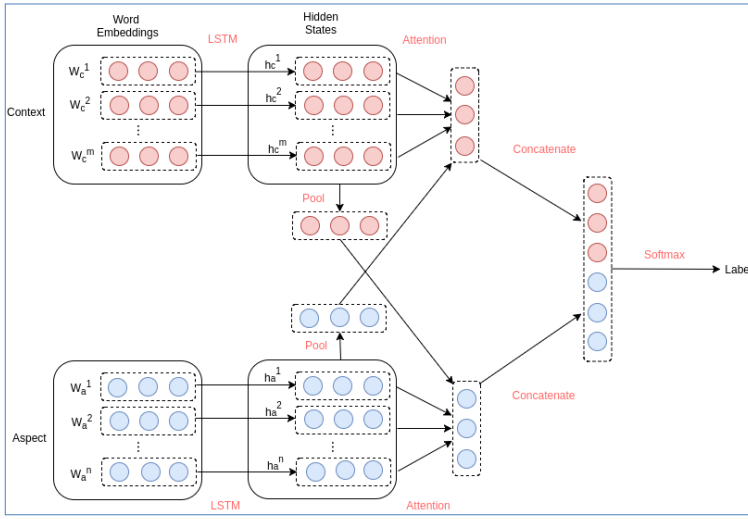

Рис. 2.Архитектура модели IAN
Fig. 2. The overall architecture of IAN

4.4 RAM

Сеть с рекуррентным механизмом внимания к памяти была представлена Ченом с соавторами [9]. Сеть состоит из трех главных частей: первая посвящена обработке контекста с использованием двунаправленной LSTM, полученные вектора сохраняются в память; вторая отвечает за представление классифицируемой сущности и также использует двунаправленную LSTM, на выходе получается среднее значение всех векторов скрытого состояния слов сущности; третья часть применяет механизмы внимания к полученным выходным данным второй части и сохраненным данным первой части. Выходной вектор внимания подается на вход слою с управляемыми рекуррентными блоками (Gated Recurrent Unit; GRU). На следующей итерации на вход слою с вниманием подается выход из GRU и вектора, сохраненные в

памяти. Это позволяет применить механизм внимания к сохраненным в память данным несколько раз и извлечь больше необходимой для классификации информации. Вектор, полученный в результате нескольких подобных итераций, передается в полносвязный слой с классификатором. Архитектура RAM представлена на рис. 3.

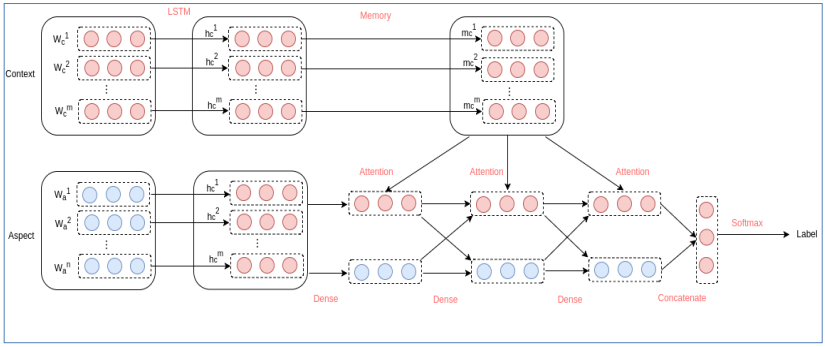

Рис. 3. Архитектура модели RAM
Fig. 3. The overall architecture of RAM

4.5 MemNet

Модель MemNet была представлена Тангом с соавторами [8]. Данная модель состоит из двух главных частей: модуля памяти, который хранит в себе входные данные для контекста в виде распределенного представления слов и механизма внимания. На вход слою с вниманием подаются сущность в виде векторного представления слов и вектора, сохраненные в памяти. Выход из слоя памяти суммируется с векторами памяти и подается в следующий слой с механизмом внимания. Архитектура RAM представлена на рис. 4.

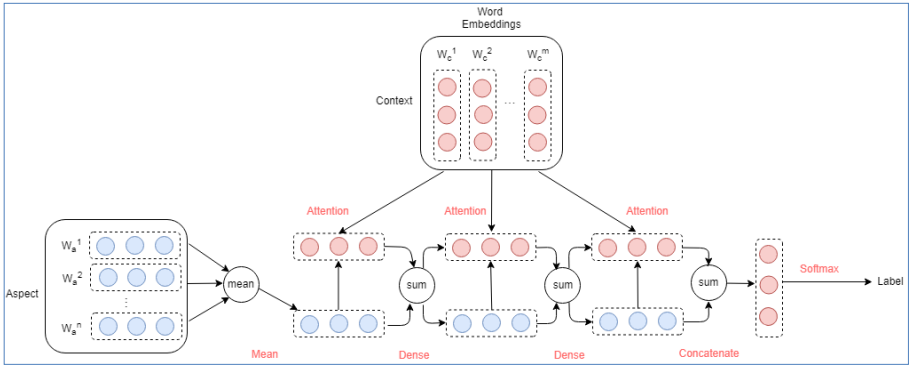

Рис. 4. Архитектура модели MemNet
Fig. 4. The overall architecture of MemNet

5. Эксперименты

В этом разделе мы представляем сравнение эффективности описанных нейронных сетей с моделью на основе метода опорных векторов (Support vector Machine; SVM) с большим набором признаков, чтобы ответить на ключевые вопросы исследования, изложенные во введении.

5.1 Метод на основе SVM

Мы сравнили наши подходы с классификатором предложенном в работе [37]. Данный метод основан на SVM с линейным ядром. Набор экспериментов показал, что признаки на основе униграмм, бинрамм, частей речи, тональности, векторов кластера и семантических типов из словаря UMLS являются наиболее эффективными для классификации побочных эффектов. Признак на основе частей речи состоит и количества существительных, глаголов, наречий и прилагательных. Для тонального признака использовались словари: SentiWordNet [46], MPQA Subjectivity Lexicon [45], Bing Liu's словарь [47]. Признак на основе кластерного представления использовал кластера из [38], полученные с использованием иерархического алгоритма кластеризации Брауна. Последний признак представляет собой количество токенов из каждого семантического типа словаря UMLS.

Оценка эффективности данного метода показала его превосходство в сравнении с предыдущими подходами, основанными на методах машинного обучения и сверточной нейронной сети.

5.2 Параметры моделей

Мы использовали векторное представление слов, обученное на записях из социальных медиа [38]. Векторное представление слов было получено с использованием модели word2vec, обученной на неразмеченном корпусе, состоящем из 2.5 миллиона англоязычных отзывов пользователей о лекарственных препаратах. Длина векторов 200. Статистика покрываемости корпусов словами из модели векторного представления слов: CADEC - 93.5%, Twitter - 80.4%, MADE - 62.5%, TwiMed-Twitter - 81.2%, TwiMed-Pubmed - 76.4%. Для слов, отсутствующих в модели, генерируется вектор случайных чисел с нормальным распределением и значениями, ранжирующимися в рамках значений векторов модели векторного представления слов. Мы использовали 15 эпох для обучения каждой модели на каждом из корпусов, размер входного блока 128 для корпусов CADEC и MADE и 32 для остальных корпусов, количество скрытых состояний 300, шаг обучения (learning rate) 0.01, l2 регуляризация со значением 0.001. В ходе экспериментов модель с данным набором параметров показала наиболее высокий результат. Для

реализации модели был использован публично доступный код из репозитория[1].

5.3 Результаты

Все модели были оценены на 5-фолдовой кросс валидации с помощью стандартных метрик оценки качества классификации: точность (P), полнота (R) и F-мера – среднее гармоническое между точностью и полнотой. Результаты экспериментов приведены в табл. 2-6. В таблицах класс ‘ADR’ обозначает класс с побочным эффектом, соответственно, класс ‘non-ADR’ обозначает его отсутствие.

Из результатов видно, что на всех корпусах, кроме Twitter лучшие результаты по макро F-мере показала модель IAN. Наиболее значимый прирост качества по сравнению с другими моделями был получен на корпусах Twimed-Twitter и Twitter-Pubmed, где модель IAN достигла 81.9% и 87.4% макро F-меры соответственно. На корпусе Twitter лучшие результаты показала модель RAM с макро F-мерой 83.4%.

Исходя из полученных результатов, можно сделать вывод, что разделение входного предложения на правый и левый контекст относительно выделенной сущности может улучшить качество классификации для корпусов, состоящих из твитов. Это следует из того, что TD_LSTM с результатами макро F-меры 75.8% и 70.3% на корпусах Twitter и Twimed-Twitter соответственно превзошли модель LSTM с результатами 61.3% и 70% макро F-меры. Для остальных корпусов разделение контекста не смогло улучшить результатов. На корпусе Twimed-Pubmed LSTM превзошла модель TD_LSTM на 7% по метрике F-меры. На остальных корпусах результаты сравнимы и отличаются всего на 2%.

Сравнение результатов работы моделей RAM и MemNet показывают, что наличие LSTM слоя перед слоем с памяти оказалось эффективно только на одном корпусе Twitter, где RAM показала существенно высокие результаты F-меры (83.4%) по сравнению с MemNet (76.3%).

Превосходство IAN по сравнению с RAM и MemNet на четырех из пяти корпусов также показывает, что наличие дополнительной памяти далеко не всегда дает преимущество.

Табл. 2. Результаты классификации на корпусе Twitter
Tab. 2. Classification results of the compared methods for Twitter corpus

Модель	Класс non-ADR			Класс ADR			Макро		
	P	R	F	P	R	F	P	R	F
SVM	.602	.520	.554	.602	.520	.554	.769	.736	.749

[1] https://github.com/songyouwei/ABSA-PyTorch

IAN	.654	.627	.634	.951	.957	.954	.802	.792	.794
RAM	**.779**	.653	**.705**	**.955**	**.973**	**.964**	**.867**	**.813**	**.834**
MemNet	.559	**.667**	.590	.954	.918	.935	.757	.792	.763
TD-LSTM	.606	.547	.570	.940	.952	.946	.773	.749	.758
LSTM	.388	.427	.392	.920	.889	.903	.618	.621	.613

Табл. 3. Результаты классификации на корпусе CADEC
Tab. 3. Classification results of the compared methods for CADEC corpus

Модель	Класс non-ADR			Класс ADR			Макро		
	P	R	F	P	R	F	P	R	F
SVM	.659	.620	.638	.964	.969	.967	.811	.795	.802
IAN	**.699**	**.637**	**.662**	**.966**	.972	**.969**	**.832**	**.805**	**.815**
RAM	.696	.406	.506	.946	**.981**	.963	.821	.694	.734
MemNet	.575	.570	.559	.960	.955	.957	.767	.762	.758
TD-LSTM	.630	.557	.582	.958	.967	.962	.794	.762	.772
LSTM	.664	.554	.602	.958	.973	.966	.811	.764	.784

Табл. 4. Результаты классификации на корпусе MADE
Tab. 4. Classification results of the compared methods for MADE corpus

Модель	Класс non-ADR			Класс ADR			Макро		
	P	R	F	P	R	F	P	R	F
SVM	**.984**	.981	.982	.551	**.582**	.562	.767	**.782**	.772
IAN	.982	**.991**	**.986**	**.740**	.524	**.585**	**.861**	.758	**.786**
RAM	.980	.989	.985	.615	.486	.538	.798	.737	.761
MemNet	.979	**.991**	.985	.684	.447	.535	.832	.719	.760
TD-LSTM	.980	.988	.984	.606	.470	.515	.793	.729	.750
LSTM	.981	.989	.985	.636	.510	.557	.809	.749	.771

Табл. 5. Результаты классификации на корпусе Twimed-Twitter
Tab. 5. Classification results of the compared methods for Twimed-Twitter corpus

Модель	Класс non-ADR			Класс ADR			Макро		
	P	R	F	P	R	F	P	R	F
SVM	.779	.707	.739	**.752**	.810	.778	.766	.758	.758

IAN	**.802**	**.825**	**.813**	**.836**	.813	**.824**	**.819**	**.819**	**.819**
RAM	.799	.736	.764	.773	**.823**	.796	.786	.779	.780
MemNet	.772	.821	.789	.823	.791	.801	.798	.806	.795
TD-LSTM	.731	.711	.717	.741	.751	.742	.736	.731	.730
LSTM	.669	.757	.709	.743	.649	.691	.706	.703	.700

Табл. 6. Результаты классификации на корпусе Twimed-Pubmed
Tab. 6. Classification results of the compared methods for Twimed-Pubmed corpus

Модель	Класс non-ADR			Класс ADR			Макро		
	P	R	F	P	R	F	P	R	F
SVM	.925	.955	0.939	.799	.681	.728	.862	.818	.834
IAN	**.936**	**.977**	**.956**	.878	.738	0.792	**.907**	**.858**	**.874**
RAM	.917	.916	.916	.675	.669	0.662	.796	.792	.789
MemNet	.929	.912	.917	.736	.748	0.705	.833	.830	.811
TD-LSTM	.495	.493	.487	**.932**	**.930**	**0.931**	.714	.712	.709
LSTM	.929	.949	.939	.786	.707	0.740	.858	.828	.839

6 Заключение

В данной статье была исследована применимость общепринятых архитектур нейронных сетей в области аспектно-ориентированного анализа тональности к задаче классификации побочных эффектов. Для оценки эффективности данных моделей были проведены обширные эксперименты на пяти общедоступных текстовых корпусах. Согласно полученным результатам, для четырех из пяти корпусов наилучшие результаты показала модель IAN и на одном корпусе RAM. Также можно сделать вывод, что базовые архитектуры не уступают результатам работы существующего метода на основе SVM, а сети с дополнительной памятью и механизмом внимания превосходят их, что доказывает применимость данных архитектур к задаче классификации побочных эффектов.

В дальнейшем планируются исследования по трем направлениям:

1) оценка влияние параметров предложенных архитектур нейронных сетей на качество классификации;
2) адаптация описанных моделей для классификации на уровне сообщений;
3) применение данных моделей для задачи классификации побочных эффектов на других языках.

7 Благодарности

Работа поддержана грантом РНФ № 18-11-00284.

Список литературы

[1]. Murff HJ, Patel VL, Hripcsak G, Bates DW. Detecting adverse events for patient safety research: a review of current methodologies. Journal of Biomedical Informatics, vol. 36, issue ½, 2003, pp. 131–143.
[2]. Sarker A, Ginn R, Nikfarjam A, O'Connor K, Smith K, Jayaraman S et al. Utilizing social media data for pharmacovigilance: A review. Journal of Biomedical Informatics, vol. 54, 2015, pp. 202–212.
[3]. Lardon J, Abdellaoui R, Bellet F, Asfari H, Souvignet J, Texier N et al. Adverse Drug Reaction Identification and Extraction in Social Media: A Scoping Review. Journal of Medical Internet Research, vol 17, no 7, 2015.
[4]. Harpaz R, Callahan A, Tamang S, Low Y, Odgers D, Finlayson S, et al. Text mining for adverse drug events: the promise, challenges, and state of the art. Drug Safety, vol. 37, 2014, pp. 777–790.
[5]. Harpaz R, DuMouchel W, Shah NH, Madigan D, Ryan P, Friedman C. Novel data-mining methodologies for adverse drug event discovery and analysis. Clinical Pharmacology & Therapeutics, vol. 91, no. 6, 2012, pp. 1010–1021.
[6]. Tang D, Qin B, Feng X, Liu T. Effective LSTMs for Target-Dependent Sentiment Classification [Internet]. arXiv [cs.CL], 2015. Available at: http://arxiv.org/abs/1512.01100, accessed 15.11.2008
[7]. Ma D, Li S, Zhang X, Wang H. Interactive attention networks for aspect-level sentiment classification. arXiv preprint arXiv:1709 00893, 2017.
[8]. Tang D, Qin B, Liu T. Aspect level sentiment classification with deep memory network. arXiv preprint arXiv:1605 08900, 2016;
[9]. Chen P, Sun Z, Bing L, Yang W. Recurrent attention network on memory for aspect sentiment analysis. In Proc. of the 2017 Conference on Empirical Methods in Natural Language Processing. 2017, pp. 452–461.
[10]. Benton A, Ungar L, Hill S, Hennessy S, Mao J, Chung A, et al. Identifying potential adverse effects using the web: a new approach to medical hypothesis generation. Journal of Biomedical Informatics, vol. 44, 2011, pp. 989–996.
[11]. Yang CC, Yang H, Jiang L, Zhang M. Social Media Mining for Drug Safety Signal Detection. In Proc. of the 2012 International Workshop on Smart Health and Wellbeing, 2012. pp. 33–40.
[12]. Liu X, Chen H. AZDrugMiner: An Information Extraction System for Mining Patient-Reported Adverse Drug Events in Online Patient Forums / Lecture Notes in Computer Science, vol. 8040, 2013. pp. 134–150.
[13]. Yeleswarapu S, Rao A, Joseph T, Saipradeep VG, Srinivasan R. A pipeline to extract drug-adverse event pairs from multiple data sources. BMC Medical Informatics and Decision Making, vol. 14, no. 13, 2014.
[14]. Freifeld CC, Brownstein JS, Menone CM, Bao W, Filice R, Kass-Hout T, et al. Digital drug safety surveillance: monitoring pharmaceutical products in twitter. Drug Safety, vol. 37, 2014, pp. 343–350.
[15]. O'Connor K, Pimpalkhute P, Nikfarjam A, Ginn R, Smith KL, Gonzalez G. Pharmacovigilance on twitter? Mining tweets for adverse drug reactions. In Proc. of the AMIA Annual Symposium, 2014, pp. 924–933.

[16]. Nikfarjam A, Gonzalez GH. Pattern mining for extraction of mentions of Adverse Drug Reactions from user comments. In Proc. of the AMIA Annual Symposium, 2011, pp. 1019–1026.
[17]. Na J-C, Kyaing WYM, Khoo CSG, Foo S, Chang Y-K, Theng Y-L. Sentiment Classification of Drug Reviews Using a Rule-Based Linguistic Approach. Lecture Notes in Computer Science, vol. 7634, 2012. pp. 189–198.
[18]. Yun Niu et al. Analysis of polarity information in medical text. In Proc. of the AMIA Annual Symposium, 2005, pp. 570–574.
[19]. Leaman R. et al. Towards internet-age pharmacovigilance: extracting adverse drug reactions from user posts to health-related social networks. In Proc. of the 2010 workshop on biomedical natural language processing, 2010, pp. 117-125.
[20]. Yun Niu, Xiaodan Zhu et al. Predicting adverse drug events from personal health messages. In Proc. of the AMIA Annual Symposium, 2011, pp. 217-226.
[21]. Bian J., Topaloglu U., Yu F. Towards large-scale twitter mining for drug-related adverse events. In. Proc. of the 2012 International workshop on smart health and wellbeing, 2012, pp. 25-32.
[22]. Yang M., Wang X., Kiang M. Y. Identification of Consumer Adverse Drug Reaction Messages on Social Media. In Proc. of the Pacific Asia Conference on Information Systems, 2013.
[23]. Sarker A., Gonzalez G. Portable automatic text classification for adverse drug reaction detection via multi-corpus training. Journal of biomedical informatics, vol. 53, 2015, pp. 196-207.
[24]. Aramaki E. et al. Extraction of adverse drug effects from clinical records. Studies in Health Technology and Informatics, vol. 160, №. Pt 1, 2010, pp. 739-743.
[25]. Rastegar-Mojarad M., Elayavilli R.K., Yu Y., Liu H. Detecting signals in noisy data-can ensemble classifiers help identify adverse drug reaction in tweets. In Proc. of the Social Media Mining Shared Task Workshop at the Pacific Symposium on Biocomputing, 2016.
[26]. Sarker A, Nikfarjam A, Gonzalez G. Social Media Mining Shared Task Workshop. In Proc. of the Pacific Symposium on Biocomputing, 2016, pp. 581–592.
[27]. Sarker A, Gonzalez-Hernandez G. Overview of the Second Social Media Mining for Health (SMM4H) Shared Tasks at AMIA 2017. In Proc. of the 2nd Social Media Mining for Health Research and Applications Workshop, 2017, pp. 43-48.
[28]. Kiritchenko S, Mohammad SM, Morin J, de Bruijn B. NRC-Canada at SMM4H Shared Task: Classifying Tweets Mentioning Adverse Drug Reactions and Medication Intake. arXiv preprint arXiv:1805 04558. 2018.
[29]. Friedrichs J, Mahata D, Gupta S. InfyNLP at SMM4H Task 2: Stacked Ensemble of Shallow Convolutional Neural Networks for Identifying Personal Medication Intake from Twitter. arXiv preprint arXiv:1803 07718. 2018.
[30]. Huynh T, He Y, Willis A, Rüger S. Adverse drug reaction classification with deep neural networks. In Proc. of the 26th International Conference on Computational Linguistics: Technical Papers, 2016, pp. 877–887.
[31]. Gurulingappa H., Rajput A.M. et al. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. Journal of Biomedical Informatics, vol. 45, 2012, pp. 885–892.
[32]. Serrano-Guerrero J., Olivas J.A. et al. Sentiment analysis: A review and comparative analysis of web services. Information Sciences, vol. 311, 2015, pp. 18–38

[33]. Rusnachenko N., Loukachevitch N. Using convolutional neural networks for sentiment attitude extraction from analytical texts. In Proc. of the Third Workshop on Computational linguistics and language science (to be published in CEUR Workshop Proceedings), 2018

[34]. Ivanov V., Tutubalina E., Mingazov N., Alimova I. Extracting aspects, sentiment and categories of aspects in user reviews about restaurants and cars. Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference "Dialogue", issue 14, vol. 2, 2015, pp. 22–34

[35]. Solovyev V., Ivanov V. Dictionary-based problem phrase extraction from user reviews. Lecture Notes in Computer Science, vol. 8655, 2014, pp. 225–232.

[36]. Zhang L., Wang S., Liu, B. Deep learning for sentiment analysis. A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 8, issue 4, 2018.

[37]. Alimova I., Tutubalina E. Automated detection of adverse drug reactions from social media posts with machine learning. Lecture Notes in Computer, vol. 10716, 2017, pp. 3–15.

[38]. Miftahutdinov Z.S., Tutubalina E.V., Tropsha A.E. Identifying disease-related expressions in reviews using conditional random fields. Computational Linguistics and Intellectual Technologies: Papers from the Annual conference "Dialogue", issue 16, vol. 1, 2017, pp 155–166

[39]. Korkontzelos I., Nikfarjam A. et al. Analysis of the effect of sentiment analysis on extracting adverse drug reactions from tweets and forum posts. Journal of biomedical informatics, vol. 62, 2016, pp. 148–158.

[40]. Dai H.-J., Touray M., Jonnagaddala J., Syed-Abdul S. Feature engineering for recognizing adverse drug reactions from twitter posts. Information, vol. 7, no. 27, 2016.

[41]. Karimi, S. Metke-Jimenez,A., Kemp M., Wang C.: Cadec. A corpus of adverse drug event annotations. Journal of biomedical informatics, vol. 55, 2015, pp. 73–81.

[42]. Nikfarjam A., Sarker A. et al. Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. Journal of the American Medical Informatics Association, vol. 22, no. 3, 2015, pp. 671–681

[43]. Nlp challenges for detecting medication and adverse drug events from electronic health records (made1.0) (2018). University of Massachusetts Lowell, Worcester, Amhers. Available at: https://bio-nlp.org/index.php/projects/39-nlp-challenges, accessed 15.11.2008.

[44]. Alvaro N., Miyao Y., Collier N. Twimed: Twitter and pubmed comparable corpus of drugs, diseases, symptoms, and their relations. JMIR public health and surveillance, vol. 3, no. 2, 2017.

[45]. Wilson T., Wiebe J., Hoffmann P. Recognizing contextual polarity in phrase-level sentiment analysis. In Proc. of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, 2005, pp. 347–

[46]. Baccianella S., Esuli A., Sebastiani F. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In Proceedings of the Seventh conference on International Language Resources and Evaluation, 2010, pp. 2200–2204 (2010)

[47]. Hu M., Liu B. Mining and summarizing customer reviews. In Proc, of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 168–177.

Entity-level classification of adverse drug reactions: a comparison of neural network models

I.S. Alimova <alimovailseyar@gmail.com>
E.V. Tutubalina <tutubalinaev@gmail.com >
Kazan Federal University,
18 Kremlyovskaya street, Kazan, 420008, Russian Federation

Abstract. This paper presents our experimental work on neural network models for entity-level adverse drug reaction (ADR) classification. Aspect-level sentiment classification, which aims to determine the sentimental class of a specific aspect conveyed in user opinions, have been actively studied for more than 10 years. In the past few years, several neural network models have been proposed to address this problem. While these models have a lot in common, there are some architecture components that distinguish them from each other. We investigate the applicability of neural network models for ADR classification. We conduct extensive experiments on various pharmacovigilance text sources including biomedical literature, clinical narratives, and social media and compare the performance of five state-of-the-art models as well as a feature-rich SVM in terms of the accuracy of ADR classification.

Keywords: adverse drug reactions; text mining; natural language processing; health social media analytics; machine learning; deep learning

DOI: 10.15514/ISPRAS-2018-30(5)-11

For citation: Alimova I.S., Tutubalina E.V. Entity-level classification of adverse drug reactions: a comparison of neural network models. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 177-196 (in Russian). DOI: 10.15514/ISPRAS-2018-30(5)-11

References

[1]. Murff HJ, Patel VL, Hripcsak G, Bates DW. Detecting adverse events for patient safety research: a review of current methodologies. Journal of Biomedical Informatics, vol. 36, issue ½, 2003, pp. 131–143.
[2]. Sarker A, Ginn R, Nikfarjam A, O'Connor K, Smith K, Jayaraman S et al. Utilizing social media data for pharmacovigilance: A review. Journal of Biomedical Informatics, vol. 54, 2015, pp. 202–212.
[3]. Lardon J, Abdellaoui R, Bellet F, Asfari H, Souvignet J, Texier N et al. Adverse Drug Reaction Identification and Extraction in Social Media: A Scoping Review. Journal of Medical Internet Research, vol 17, no 7, 2015.
[4]. Harpaz R, Callahan A, Tamang S, Low Y, Odgers D, Finlayson S, et al. Text mining for adverse drug events: the promise, challenges, and state of the art. Drug Safety, vol. 37, 2014, pp. 777–790.
[5]. Harpaz R, DuMouchel W, Shah NH, Madigan D, Ryan P, Friedman C. Novel data-mining methodologies for adverse drug event discovery and analysis. Clinical Pharmacology & Therapeutics, vol. 91, no. 6, 2012, pp. 1010–1021.
[6]. Tang D, Qin B, Feng X, Liu T. Effective LSTMs for Target-Dependent Sentiment Classification [Internet]. arXiv [cs.CL], 2015. Available at: http://arxiv.org/abs/1512.01100, accessed 15.11.2008

[7]. Ma D, Li S, Zhang X, Wang H. Interactive attention networks for aspect-level sentiment classification. arXiv preprint arXiv:1709 00893, 2017.
[8]. Tang D, Qin B, Liu T. Aspect level sentiment classification with deep memory network. arXiv preprint arXiv:1605 08900, 2016;
[9]. Chen P, Sun Z, Bing L, Yang W. Recurrent attention network on memory for aspect sentiment analysis. In Proc. of the 2017 Conference on Empirical Methods in Natural Language Processing. 2017, pp. 452–461.
[10]. Benton A, Ungar L, Hill S, Hennessy S, Mao J, Chung A, et al. Identifying potential adverse effects using the web: a new approach to medical hypothesis generation. Journal of Biomedical Informatics, vol. 44, 2011, pp. 989–996.
[11]. Yang CC, Yang H, Jiang L, Zhang M. Social Media Mining for Drug Safety Signal Detection. In Proc. of the 2012 International Workshop on Smart Health and Wellbeing, 2012. pp. 33–40.
[12]. Liu X, Chen H. AZDrugMiner: An Information Extraction System for Mining Patient-Reported Adverse Drug Events in Online Patient Forums / Lecture Notes in Computer Science, vol. 8040, 2013. pp. 134–150.
[13]. Yeleswarapu S, Rao A, Joseph T, Saipradeep VG, Srinivasan R. A pipeline to extract drug-adverse event pairs from multiple data sources. BMC Medical Informatics and Decision Making, vol. 14, no. 13, 2014.
[14]. Freifeld CC, Brownstein JS, Menone CM, Bao W, Filice R, Kass-Hout T, et al. Digital drug safety surveillance: monitoring pharmaceutical products in twitter. Drug Safety, vol. 37, 2014, pp. 343–350.
[15]. O'Connor K, Pimpalkhute P, Nikfarjam A, Ginn R, Smith KL, Gonzalez G. Pharmacovigilance on twitter? Mining tweets for adverse drug reactions. In Proc. of the AMIA Annual Symposium, 2014, pp. 924–933.
[16]. Nikfarjam A, Gonzalez GH. Pattern mining for extraction of mentions of Adverse Drug Reactions from user comments. In Proc. of the AMIA Annual Symposium, 2011, pp. 1019–1026.
[17]. Na J-C, Kyaing WYM, Khoo CSG, Foo S, Chang Y-K, Theng Y-L. Sentiment Classification of Drug Reviews Using a Rule-Based Linguistic Approach. Lecture Notes in Computer Science, vol. 7634, 2012. pp. 189–198.
[18]. Yun Niu et al. Analysis of polarity information in medical text. In Proc. of the AMIA Annual Symposium, 2005, pp. 570–574.
[19]. Leaman R. et al. Towards internet-age pharmacovigilance: extracting adverse drug reactions from user posts to health-related social networks. In Proc. of the 2010 workshop on biomedical natural language processing, 2010, pp. 117-125.
[20]. Yun Niu, Xiaodan Zhu et al. Predicting adverse drug events from personal health messages. In Proc. of the AMIA Annual Symposium, 2011, pp. 217-226.
[21]. Bian J., Topaloglu U., Yu F. Towards large-scale twitter mining for drug-related adverse events. In. Proc. of the 2012 International workshop on smart health and wellbeing, 2012, pp. 25-32.
[22]. Yang M., Wang X., Kiang M. Y. Identification of Consumer Adverse Drug Reaction Messages on Social Media. In Proc. of the Pacific Asia Conference on Information Systems, 2013.
[23]. Sarker A., Gonzalez G. Portable automatic text classification for adverse drug reaction detection via multi-corpus training. Journal of biomedical informatics, vol. 53, 2015, pp. 196-207.

[24]. Aramaki E. et al. Extraction of adverse drug effects from clinical records. Studies in Health Technology and Informatics, vol. 160, №. Pt 1, 2010, pp. 739-743.

[25]. Rastegar-Mojarad M., Elayavilli R.K., Yu Y., Liu H. Detecting signals in noisy data-can ensemble classifiers help identify adverse drug reaction in tweets. In Proc. of the Social Media Mining Shared Task Workshop at the Pacific Symposium on Biocomputing, 2016.

[26]. Sarker A, Nikfarjam A, Gonzalez G. Social Media Mining Shared Task Workshop. In Proc. of the Pacific Symposium on Biocomputing, 2016, pp. 581–592.

[27]. Sarker A, Gonzalez-Hernandez G. Overview of the Second Social Media Mining for Health (SMM4H) Shared Tasks at AMIA 2017. In Proc. of the 2nd Social Media Mining for Health Research and Applications Workshop, 2017, pp. 43-48.

[28]. Kiritchenko S, Mohammad SM, Morin J, de Bruijn B. NRC-Canada at SMM4H Shared Task: Classifying Tweets Mentioning Adverse Drug Reactions and Medication Intake. arXiv preprint arXiv:1805 04558. 2018.

[29]. Friedrichs J, Mahata D, Gupta S. InfyNLP at SMM4H Task 2: Stacked Ensemble of Shallow Convolutional Neural Networks for Identifying Personal Medication Intake from Twitter. arXiv preprint arXiv:1803 07718. 2018.

[30]. Huynh T, He Y, Willis A, Rüger S. Adverse drug reaction classification with deep neural networks. In Proc. of the 26th International Conference on Computational Linguistics: Technical Papers, 2016, pp. 877–887.

[31]. Gurulingappa H., Rajput A.M. et al. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. Journal of Biomedical Informatics, vol. 45, 2012, pp. 885–892.

[32]. Serrano-Guerrero J., Olivas J.A. et al. Sentiment analysis: A review and comparative analysis of web services. Information Sciences, vol. 311, 2015, pp. 18–38

[33]. Rusnachenko N., Loukachevitch N. Using convolutional neural networks for sentiment attitude extraction from analytical texts. In Proc. of the Third Workshop on Computational linguistics and language science (to be published in CEUR Workshop Proceedings), 2018

[34]. Ivanov V., Tutubalina E., Mingazov N., Alimova I. Extracting aspects, sentiment and categories of aspects in user reviews about restaurants and cars. Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference “Dialogue”, issue 14, vol. 2, 2015, pp. 22–34

[35]. Solovyev V., Ivanov V. Dictionary-based problem phrase extraction from user reviews. Lecture Notes in Computer Science, vol. 8655, 2014, pp. 225–232.

[36]. Zhang L., Wang S., Liu, B. Deep learning for sentiment analysis. A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 8, issue 4, 2018.

[37]. Alimova I., Tutubalina E. Automated detection of adverse drug reactions from social media posts with machine learning. Lecture Notes in Computer, vol. 10716, 2017, pp. 3–15.

[38]. Miftahutdinov Z.S., Tutubalina E.V., Tropsha A.E. Identifying disease-related expressions in reviews using conditional random fields. Computational Linguistics and Intellectual Technologies: Papers from the Annual conference “Dialogue”, issue 16, vol. 1, 2017, pp 155–166

[39]. Korkontzelos I., Nikfarjam A. et al. Analysis of the effect of sentiment analysis on extracting adverse drug reactions from tweets and forum posts. Journal of biomedical informatics, vol. 62, 2016, pp. 148–158.

[40]. Dai H.-J., Touray M., Jonnagaddala J., Syed-Abdul S. Feature engineering for recognizing adverse drug reactions from twitter posts. Information, vol. 7, no. 27, 2016.
[41]. Karimi, S. Metke-Jimenez,A., Kemp M., Wang C.: Cadec. A corpus of adverse drug event annotations. Journal of biomedical informatics, vol. 55, 2015, pp. 73–81.
[42]. Nikfarjam A., Sarker A. et al. Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. Journal of the American Medical Informatics Association, vol. 22, no. 3, 2015, pp. 671–681
[43]. Nlp challenges for detecting medication and adverse drug events from electronic health records (made1.0) (2018). University of Massachusetts Lowell, Worcester, Amhers. Available at: https://bio-nlp.org/index.php/projects/39-nlp-challenges, accessed 15.11.2008.
[44]. Alvaro N., Miyao Y., Collier N. Twimed: Twitter and pubmed comparable corpus of drugs, diseases, symptoms, and their relations. JMIR public health and surveillance, vol. 3, no. 2, 2017.
[45]. Wilson T., Wiebe J., Hoffmann P. Recognizing contextual polarity in phrase-level sentiment analysis. In Proc. of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, 2005, pp. 347–
[46]. Baccianella S., Esuli A., Sebastiani F. Sentiwordnet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In Proceedings of the Seventh conference on International Language Resources and Evaluation, 2010, pp. 2200–2204 (2010)
[47]. Hu M., Liu B. Mining and summarizing customer reviews. In Proc, of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 168–177.

Rock Flow Simulation by High-Order Quasi-Characteristics Scheme

Mikhail P. Levin <mlevin@ispras.ru>
Ivannikov Institute for System Programming of RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. A pure second-order scheme of quasi-characteristics based on a pyramidal stencil is applied to the numerical modelling of non-stationary two-phase flows through porous media with the essentially heterogeneous properties. In contrast to well-known other high-resolution schemes with monotone properties, this scheme preserves a second-order approximation in regions, where discontinuities of solutions arise, as well as monotone properties of numerical solutions in those regions despite of well-known Godunov theorem. It is possible because the scheme under consideration is defined on a non-fixed stencil and is a combination of two high-order approximation scheme solutions with different dispersion properties. A special criterion according to which, one or another admissible solution is chosen, plays a key role in this scheme. A simple criterion with local character suitable for parallel computations is proposed. Some numerical results showing the efficiency of present approach in computations of two-phase flows through porous media with strongly discontinuous penetration coefficients are presented.

Keywords: Quasi-Characteristics; Two-phase Porous Media Flows; Heterogeneous Media

DOI: 10.15514/ISPRAS-2018-30(5)-12

For citation: Levin M.P. Rock Flow Simulation by High-Order Quasi-Characteristics Scheme. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 197-212. DOI: 10.15514/ISPRAS-2018-30(5)-12

1. Introduction

In recent fifteen years many high-resolution numerical schemes modifying Godunov scheme have been proposed (see, for instance, [1-5]). However, the problem of development of high-order schemes with monotone properties in regions near the discontinuities of solutions remains in the focus of activities for many researchers in numerical methods for partial differential equations (PDE) and in computational fluid dynamics (CFD). According to the well-known Godunov's theorem, second-order explicit monotone schemes on the fixed stencils do not exist. Up to now, two different ways to resolve this restriction are known. The first one uses the idea of lowering the approximation order in the narrow regions near the discontinuities of solution. In fact, this approach has been realized in most of the modern high resolution schemes, because they set some restrictions on the recovery functions or

limiters to provide the monotone properties of solutions in zones where the discontinuities could arise. An excellent analysis of this approach is presented in [3,5]. Therefore, most of high-resolution schemes cited above are hybrid schemes, because their approximation order is lowered in the zones near the discontinuities. Various hybrid quasi-characteristics schemes for the solution of supersonic aerodynamics problems and two-phase porous media problems were developed and considered in [6-11].

The second way consists in the construction of high-resolution schemes on the non-fixed stencils. For instance, one can apply two or more high-order schemes defined on different stencils and choose a final solution in each nodal point among admissible solutions to provide a monotone properties in regions where the discontinuities could arise. Such approach was considered in [12-14]. In these articles, various quasi-characteristics schemes of the second-order were proposed and considered. All these schemes use a combination of two second-order approximation scheme solutions having the different dispersion properties. A special criterion according to which, one or another admissible solution is chosen between two admissible solutions to provide the monotone properties near the discontinuities, plays a key role in this scheme. In [12-13], a heuristic criterion based on the third-order theoretical estimation of the average value of the governing equation operator with respect to the grid cell is proposed. Unfortunately, it has a non-local and directed character and could not be easily adopted in multi-dimensional case and in parallel computations. In [14], simpler local and non-directed heuristic criterions suitable for parallel computations are proposed. As is shown in [12], the quasi-characteristics schemes are more accurate than fourth-order approximation schemes in computing of the initial-value problems for the PDE of hyperbolic type, because the quasi-characteristics schemes are generalization of well-known back-ward characteristics schemes which are essentially more accurate in comparison with all other well-known numerical schemes. The reason of this consists in the naturally accurate treatment of the characteristic properties of the governing equations by the quasi-characteristics schemes in comparison with Godunov's type schemes based on the conservation laws treatment. Therefore, in recent years, various numerical schemes based on characteristics were proposed [15-19] for the solution of initial boundary value problems for reaction-diffusion equation and for correct setting of boundary conditions in decomposition of such problems.

In this article, we consider the application of a new multidimensional scheme of quasi-characteristics to the numerical simulation of two-phase flows through porous media with strongly discontinuous penetration coefficients. This scheme approximates a transport equation in the system of the porous media equations on the pyramidal stencil without any splitting. A simple criterion suitable for the selection of final solution among two admissible solutions to provide the monotone properties of the final solution without spuriuos oscillations is proposed. Numerical results for various ratio of penetration coefficients are presented. These results show

that the technique considered here could be efficiently used for the accurate numerical modelling of flows through the essentially heterogeneous porous media.

2. Governing equations, initial and boundary conditions

Let us consider the problem of a numerical simulation of two-phase flows through essentially heterogeneous porous medium with piece-wise constant absolute penetration factor. In the two-phase case, the governing equations [20] can be presented in the following form with respect to the water saturation s and the pressure p as unknown functions

$$m(\frac{\partial s}{\partial t}) - \frac{\partial}{\partial x}(\frac{kk_w}{\mu_w}\frac{\partial p}{\partial x}) - \frac{\partial}{\partial y}(\frac{kk_w}{\mu_w}\frac{\partial p}{\partial y}) = 0\ , \tag{1}$$

$$\frac{\partial}{\partial x}[k(\frac{k_w}{\mu_w} + \frac{k_o}{\mu_o})\frac{\partial p}{\partial x}] + \frac{\partial}{\partial y}[k(\frac{k_w}{\mu_w} + \frac{k_o}{\mu_o})\frac{\partial p}{\partial y}] = 0\ . \tag{2}$$

Here m is a porosity factor, $k = k(x, y)$ is an absolute penetration factor of porous medium, $k_w = k_w(s)$ and $k_o = k_o(s)$ are a relative penetration factors of water and oil, μ_w and μ_o are a viscosity of water and oil. Let us notice that the oil saturation s_o can be evaluated by the water saturation according to the following simple formula $s_o = 1 - s$.

Since the relative penetration factors k_w and k_o are functions only of water saturation s, then equation (1) can be presented as follows

$$\begin{gathered}\frac{\partial s}{\partial t} - (\frac{k}{m\mu_w}\frac{\partial p}{\partial x}\frac{dk_w}{ds})\frac{\partial s}{\partial x} - (\frac{k}{m\mu_w}\frac{\partial p}{\partial y}\frac{dk_w}{ds})\frac{\partial s}{\partial y} = \\ = \frac{k_w}{m}[\frac{\partial}{\partial x}(\frac{k}{\mu_w}\frac{\partial p}{\partial x}) + \frac{\partial}{\partial y}(\frac{k}{\mu_w}\frac{\partial p}{\partial y})].\end{gathered} \tag{3}$$

Now we see that the system of governing equations (23) is of mixed type. Equation (3) is a nonlinear transport equation of hyperbolic type and the equation (2) is of elliptic type. Let us consider the transport equation (3) as a main governing equation and the elliptic equation (2) as a nonlinear restriction for coefficients of the main governing equation. Then we can apply the quasi-characteristics technique to solve the initial boundary-value problem for the transport equation and also on each time level we need to solve the boundary-value problem for elliptic equation to define the coefficients of the governing equation. In our approach, for the solution of the boundary-value problem for the elliptic equation we use the well-known five points finite difference conservative scheme and bi-conjugate gradient algorithm as in [7, 11].

Let us consider rectangle flow regions $D = 0 \le x \le L,\ 0 \le y \le H$ divided into two subregions $D_1 = 0.2L \le x \le 0.8L,\ 0 \le y \le \frac{H}{2}$ and $D_2 = D - D_1$.

The absolute penetration factor k in each subregion is a constant function, therefore in all region we have

$$k = \begin{cases} k_{D_1} , & if \quad (x,y) \in D_1 , \\ k_{D_2} , & if \quad (x,y) \in D_2 . \end{cases} \tag{4}$$

Initial conditions for the transport equation (3) are

$$s(x,y,0) = \begin{cases} 0.2 , & if \quad 0 \le x < L, \ 0 \le y \le H , \\ 1.0 , & if \quad x = L, \ 0 \le y \le H \end{cases} \tag{5}$$

and the boundary conditions are

$$\begin{gathered} \frac{\partial s}{\partial y} = 0 \ , \ if \quad t > 0, \ y = 0, H, \quad 0 \le x \le L \ ; \\ s(L,y,t) = 1.0 \ , \ if \quad 0 \le y \le H \ , \quad t > 0 \ . \end{gathered} \tag{6}$$

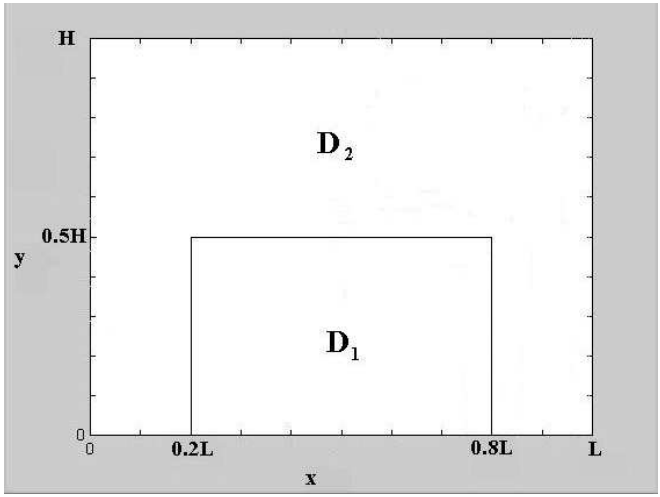

Fig.1. Flow region

For the pressure equation (2) of the elliptic type we set a mixed Neumann and Dirichlet boundary conditions as follows

$$\begin{gathered} \frac{\partial p}{\partial y} = 0 \ , \ if \quad 0 < x < L \ , \ y = 0, H; \\ p = P_0 \ , \ if \quad x = 0 \ , \ 0 \le y \le H \\ \frac{\partial p}{\partial x} = \frac{Q_0 \mu_w}{H k k_w} \ , \ if \quad x = L, \ 0 \le y \le H \ . \end{gathered} \tag{7}$$

P_0 and Q_0 are known parameters here. The relative penetration factors of the water k_w and oil k_o are chosen as follows

$$k_w(s) = \begin{cases} 0 , & if \quad s \le 0.1 \ ; \\ (\frac{s-0.1}{0.7})^3 , & if \quad 0.1 < s \le 0.8 \ ; \\ 1 , & if \quad s > 0.8 \ ; \end{cases} \tag{8}$$

$$k_o(s) = \begin{cases} 1 , & if \quad s \le 0.1 \ ; \\ (\frac{0.8-s}{0.7})^3 , & if \quad 0.1 < s \le 0.8 \ ; \\ 0 , & if \quad s > 0.8 \ . \end{cases} \tag{9}$$

Physically, the initial boundary value problem (2-9) describes two-phase porous media flows between two horizontal wells, where the left boundary $x = 0$ corresponds to the production well and the right boundary $x = L$ corresponds to the injection well.

3. Numerical scheme

In this section, we consider a non-splitting quasi-characteristics scheme on the pyramidal stencil applied to the solutions of the transport equation (3). Non-splitting scheme means that we do not use in our scheme any splitting technique for solution of the couple system of finite difference equations approximating the governing partial differential equation. It is very important in application to problems with heterogeneous coefficients, because in such problems sometimes splitting leads to the lowering of exactness of solutions. We develop this scheme with respect to the 3D transport equation written in the generalized form as follows

$$\frac{\partial u}{\partial t} + b_1 \frac{\partial u}{\partial x} + b_2 \frac{\partial u}{\partial y} = b_3, \tag{10}$$

satisfying the following initial conditions

$$u(0, x, y) = u_0(x, y) \ . \tag{11}$$

Here $u(t, x, y)$ is a searching function and $b_i = b_i(t, x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}), \quad (i = 1, 2, 3)$ and $u_0(x, y)$ are given.

In quasi-characteristics schemes [10], we approximate the governing equation written in the expanded characteristics form along some spatial grid lines l (quasi-characteristics) in (t, x, y) space as follows

$$(\frac{du}{dt})_l + [b_1 - (\frac{dx}{dt})_l]\frac{\partial u}{\partial x} + [b_2 - (\frac{dy}{dt})_l]\frac{\partial u}{\partial y} = b_3 \ . \tag{12}$$

Here $(\frac{du}{dt})_l$ is a total derivative of the searching function u with respect to t along line l.

As quasi-characteristics usually are used some grid lines belonging to the considering stencil. They should lie in close vicinity with respect to the characteristics of governing equation, and sometimes can coincide with them.

Now we consider a uniform, for simplicity, in each direction finite-difference grid in space (t, x, y). We denote grid steps τ, h_x and h_y respectively. Let us consider a pyramidal stencil $P_1P_2P_3P_4R$ in the grid space. suppose that its basement $P_1P_2P_3P_4$ belongs to some data layer $t = t_0$ and vertex R belongs to the new layer $t = t_0 + \tau$. Coordinates of the above mentioned vertices are follows: $P_1(t_0, x_0 + h_x, y_0 - h_y)$, $P_2(t_0, x_0 + h_x, y_0 + h_y)$, $P_3(t_0, x_0 - h_x, y_0 + h_y)$, $P_4(t_0, x_0 - h_x, y_0 - h_y)$, $R(t_0 + \tau, x_0, y_0)$. Also we take into consideration $m_0(t_0, x_0, y_0)$ a center point of the basement of the pyramid stencil and denote the nodal points corresponding to the central points of the pyramid basement ribs as

follows: $m_+(t_0, x_0 + h_x, y_0)$, $m_-(t_0, x_0 - h_x, y_0)$, $n_+(t_0, x_0, y_0 + h_y)$, $n_-(t_0, x_0, y_0 - h_y)$.

We suppose that the characteristics of the transport equation going through the point R is lying inside the considering stencil and as a quasi-characteristics we can choose ribs P_iR of the pyramidal stencil. Then approximating the expanded characteristic form of the governing equation along these lines, we obtain

$$\begin{array}{c} \frac{u_R - u_{P_i}}{\tau} + [(b_1)_{P_iR} - \frac{x_R - x_{P_i}}{\tau}](\frac{\partial u}{\partial x})_{P_iR} + \\ +[(b_2)_{P_iR} - \frac{y_R - y_{P_i}}{\tau}](\frac{\partial u}{\partial y})_{P_iR} = (b_3)_{P_iR}, \end{array} \tag{13}$$

where $i = 1, 2, 3, 4$.

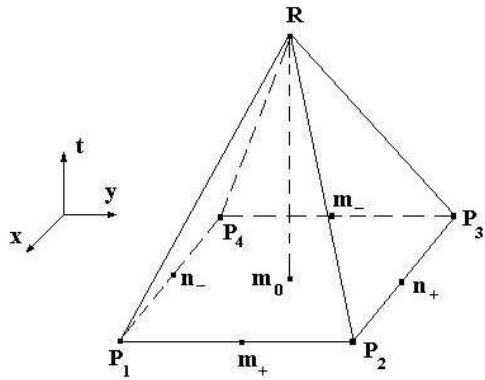

Fig.2. The pyramidal stencil

According to [10] we take the following approximation of the outward derivatives at the middle layer $t = t_0 + \frac{\tau}{2}$

$$(\frac{\partial u}{\partial x})_{t=t_0+\frac{\tau}{2}} = (\frac{\partial u}{\partial x})_C + (x - x_0)W + d(y - y_0)\ , \tag{14}$$

$$(\frac{\partial u}{\partial y})_{t=t_0+\frac{\tau}{2}} = (\frac{\partial u}{\partial y})_C + (y - y_0)V + d(x - x_0)\ . \tag{15}$$

Here we take a center point of the middle section of our stencil $(t_0 + \frac{\tau}{2}, x_0, y_0)$ as the point C (or C_I) and choose values W and V (or W_I and V_I) at the middle layer $t_0 + \frac{\tau}{2}$ by the formulas

$$\begin{array}{c} W_I = W(m_0) \equiv \frac{1}{3}[\frac{u_{P_1} - 2u_{n_-} + u_{P_4}}{{h_x}^2} \\ + \frac{u_{m_+} - 2u_{m_0} + u_{m_-}}{{h_x}^2} + \frac{u_{P_2} - 2u_{n_+} + u_{P_3}}{{h_x}^2}]\ , \end{array} \tag{16}$$

$$\begin{array}{c} V_I = V(m_0) \equiv \frac{1}{3}[\frac{u_{P_2} - 2u_{m_+} + u_{P_1}}{{h_y}^2} \\ + \frac{u_{n_+} - 2u_{m_0} + u_{n_-}}{{h_y}^2} + \frac{u_{P_3} - 2u_{m_-} + u_{P_4}}{{h_y}^2}]\ . \end{array} \tag{17}$$

We denote as $W(m_0)$ and $V(m_0)$ the finite difference operators approximating the second order derivatives of the searching function with the second order approximation error on the appropriate stencil with middle point m_0.

By substitution of relations (14-17) into (13), we obtain a system of four equations with respect to u_R, $(\frac{\partial u}{\partial x})_C$, $(\frac{\partial u}{\partial y})_C$ and d. Solving it we obtain u_R^I. In non-linear case, we need to do three iterations on nonlinear coefficients as is usually done in the method of characteristics. We call the scheme considered above **scheme I**.

Now we construct a second scheme of the second order approximation with different dispersive properties in comparison with **scheme I**. For this purpose we choose point C_{II} and values W_{II}, V_{II} according to the following formulas

$$\left.\begin{array}{l} if \;\; b_1(x_0,y_0,t_0) \geq 0 \;\; and \;\; b_2(x_0,y_0,t_0) \geq 0 \;, \\ \quad then \;\; W_{II} = W(P_4), \;\; V_{II} = V(P_4) \;, \\ \qquad C_{II} = (P_{4x}, P_{4y}, t_0 + \frac{\tau}{2}) \;, \\ if \;\; b_1(x_0,y_0,t_0) \geq 0 \;\; and \;\; b_2(x_0,y_0,t_0) < 0 \;, \\ \quad then \;\; W_{II} = W(P_3), \;\; V_{II} = V(P_3) \;, \\ \qquad C_{II} = (P_{3x}, P_{3y}, t_0 + \frac{\tau}{2}) \;, \\ if \;\; b_1(x_0,y_0,t_0) < 0 \;\; and \;\; b_2(x_0,y_0,t_0) < 0 \;, \\ \quad then \;\; W_{II} = W(P_2), \;\; V_{II} = V(P_2) \;, \\ \qquad C_{II} = (P_{2x}, P_{2y}, t_0 + \frac{\tau}{2}) \;, \\ if \;\; b_1(x_0,y_0,t_0) < 0 \;\; and \;\; b_2(x_0,y_0,t_0) \geq 0 \;, \\ \quad then \;\; W_{II} = W(P_1), \;\; V_{II} = V(P_1) \;, \\ \qquad C_{II} = (P_{1x}, P_{1y}, t_0 + \frac{\tau}{2}) \;. \end{array}\right\} \qquad (18)$$

By substitution of relations (14-15) and (18) into (13), we obtain a system of four equations with respect to u_R, $(\frac{\partial u}{\partial x})_C$, $(\frac{\partial u}{\partial y})_C$ and d. Solving it we obtain u_R^{II}. As was mentioned above in non-linear case, it is necessary to do three iterations in evaluation of u_R^{II}. We call this scheme **scheme II**.

In [12-13] for 2D case, it was shown that by choosing one of two non-monotonous admissible solutions of the second order approximation with different dispersive properties, one can construct the final solution with monotone properties. As in the papers cited before, the criterion for the choosing the final solution is based on the analysis of the average value of the governing transport equation operator evaluated on each elementary mesh cell by the high order quadrature formulas. In this criterion, the history of computations in previous grid points is taken into account and therefore is not suitable for the parallel computations and in multi-dimensional case. For 2D case, a simpler heuristic criterion based on the minimization of the rough approximation of the average value of governing operator was proposed in [14]. This criterion does not take into account the history of computations. It has a local character and it is suitable for the parallel realization.

In this paper, we construct a simple heuristic criterion as a minimal principle for the increment of searching function over the stencil in following form

$$u_R{}^{final} = \min_{i=I,II} |u_R^i - \frac{C_0 u_{m_0} + C_1 u_{P_1} + C_2 u_{P_2} + C_3 u_{P_3} + C_4 u_{P_4}}{C_0 + C_1 + C_2 + C_3 + C_4}|. \qquad (19)$$

Here $C_i,\ i = 0, 1, 2, 3, 4$ are some constants to be chosen. As our numerical tests show, the best result corresponds to the following set $C_0 = 1,\ C_i = 0,\ i = 1, 2, 3, 4$.

Thus the final solution in each grid point is chosen among two admissible solutions u_R^I and u_R^{II} according to the following simple minimal principle

$$u_R{}^{final} = \min_{i=I,II} |u_R^i - u_{m_0}|. \tag{20}$$

It is easy to see that this principle has a local character and it is very suitable for parallel computations, because it allows in principle to provide computations of searching function in each grid node independently in separate processors in computers with massive parallel processors and in computers with pipe-line processors it allows to provide the maximal loading of pipe-line.

4. Results of computations

Now let us consider some numerical results obtained by the proposed method. Computations were carried out for the following values of parameters $m = 0.2$, $k_{D_2} = 1.0 \cdot 10^{-12}\ m^2$, $\mu_w = 1 \cdot 10^{-6}\ N \cdot sec \cdot m^{-2}$, $\mu_o = 3 \cdot 10^{-6}\ N \cdot sec \cdot m^{-2}$, $L = H = 100\ m$, $P_0 = 0$, $Q_0 = 0.69444 \cdot 10^{-12}\ m^2 \cdot sec^{-1}$. Parameter k_{D_1} varies in the range from $0.50 \cdot 10^{-12}\ m^2$ to $0.01 \cdot 10^{-12}\ m^2$. Thus the absolute penetration in the subregion D_1 is 2 to 100 times less than those in subregion D_2. Presented results correspond to the uniform grid with 61*61 nodal points in (x,y)-space.

The first series of results corresponds to $k_{D_1} = 0.5 \cdot 10^{-12}\ m^2$. Fig.3 shows the isolines of water saturation s and appropriate 3D chart for time $t = 400$ hours. Fig.4 and 5 show the same results for $t = 800$ hours and $t = 1200$ hours respectively.

Fig.6 shows two functions characterizing the efficiency of the oil recovery process by the water drive. Line 1 corresponds to the ratio of the recovery oil to the total oil volume in initial moment $t = 0$ with respect to time

$$\theta(t) = \frac{\int\limits_D [1 - s(x, y, t)]dxdy}{\int\limits_D [1 - s(x, y, 0)]dxdy} \tag{21}$$

and line 2 corresponds to the function

$$\gamma(t) = \frac{\int\limits_0^L [k\frac{k_w}{\mu_w}\frac{\partial p}{\partial x}]_{x=0}dy}{\int\limits_0^L [k(\frac{k_w}{\mu_w} + \frac{k_o}{\mu_o})\frac{\partial p}{\partial x}]_{x=0}dy} \tag{22}$$

describing the water content in the development mixture at the production well corresponding to the boundary $x = 0$. According to the presented results we can see that the solution of the considering problem has a wave type and the front of water wave solution is spreading faster in the upper part of the flow region D_2 with high penetration. Subregion D_1 with low penetration plays the role of the partial obstacle and the water wave also is spreading in those region but more slowly. The second series of results corresponds to $k_{D_1} = 0.20 \cdot 10^{-12}\ m^2$ and the appropriate results are presented on Fig.7-10. In this case, we can see that there are two shock-type water waves in the considering flow. The first wave corresponds to isolines 0.25 and 0.35 and the second corresponds to 0.45 and 0.55. According to the presented results it is easy to see that the first shock wave is spreading through the region with the low penetration, but the second wave stays near the right border of the low penetration subregion.

The third series of results corresponds to $k_{D_1} = 0.01 \cdot 10^{-12}\ m^2$ and the appropriate results are presented on fig. 11-14. In this case, the water is not spreading through the region with the low penetration and the water wave front is stopping near the right border of the low penetration subregion, which plays a role of a solid obstacle in the flow region.

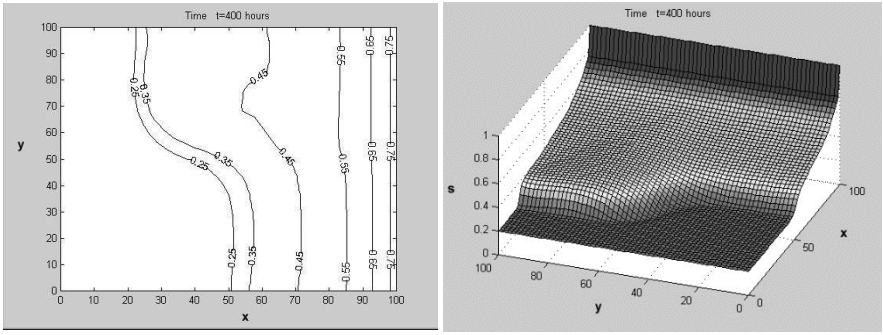

Fig.3. The water saturation at $t = 400$ *hours.* $k_{D_1} = 0.50 \cdot 10^{-12}\ m^2$.

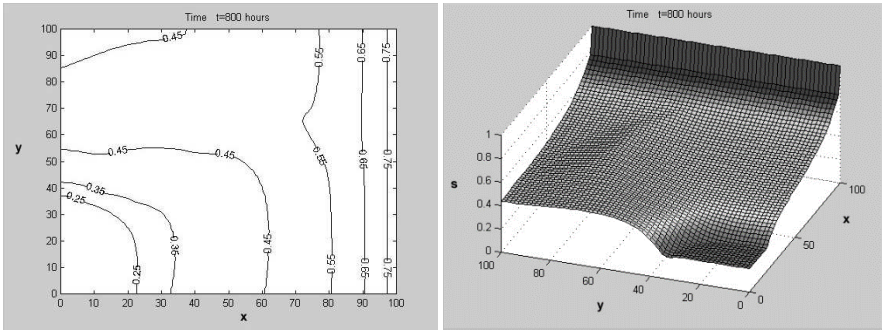

Fig.4. The water saturation at $t = 800$ *hours.* $k_{D_1} = 0.50 \cdot 10^{-12}\ m^2$.

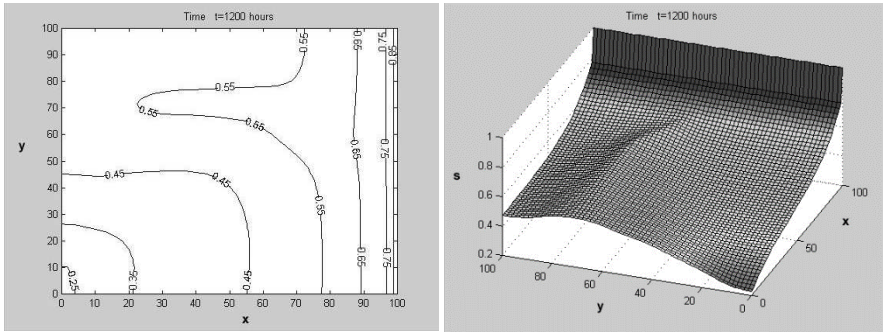

Fig.5. The water saturation at $t = 1200$ *hours.* $k_{D_1} = 0.50 \cdot 10^{-12}\ m^2$.

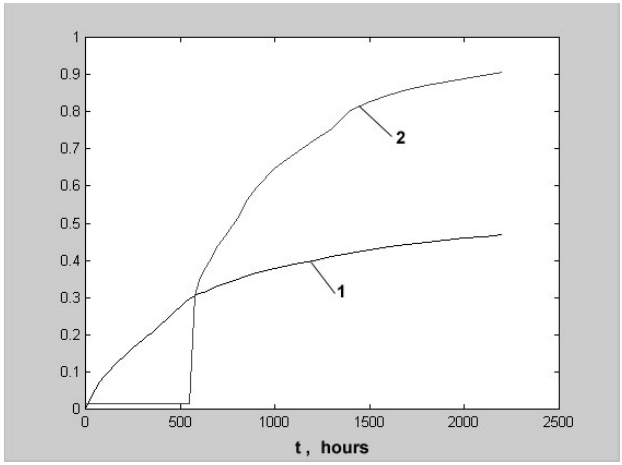

Fig.6. Characteristics of efficiency of oil recovery.
Line 1 - $\theta(t)$*, line 2 -* $\gamma(t)$*.* $k_{D_1} = 0.50 \cdot 10^{-12}\ m^2$.

The analysis of the efficiency of the oil recovery process shows that after the moment $t = 250$ hours, when the water wave in the upper part of the flow region is close to the production well (boundary $x = 0$), the efficiency falls down and oil from the low penetration subregion and even from the high penetration subregion $(0 \le x \le 0.2L,\ 0 \le y \le 0.5H)$ almost can not be developed by the water drive.

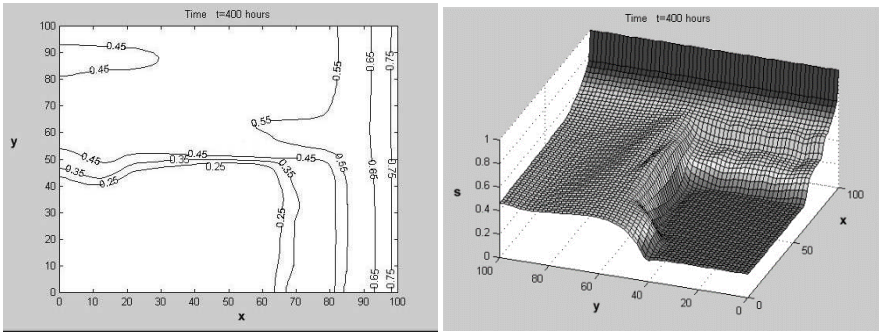

Fig.7. The water saturation at $t = 400$ *hours.* $k_{D_1} = 0.20 \cdot 10^{-12}\ m^2$.

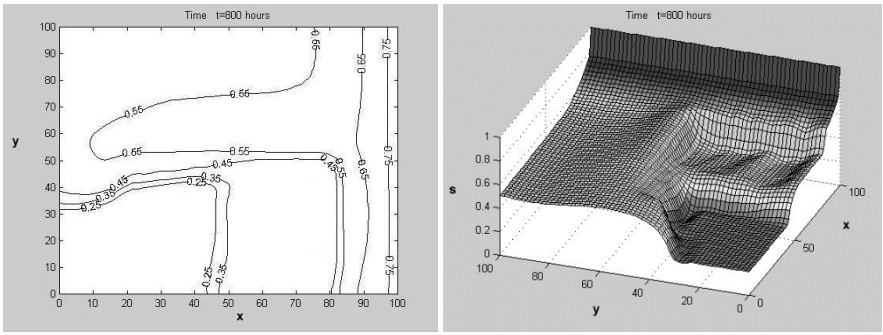

Fig.8. The water saturation at $t = 800$ *hours.* $k_{D_1} = 0.20 \cdot 10^{-12}\ m^2$.

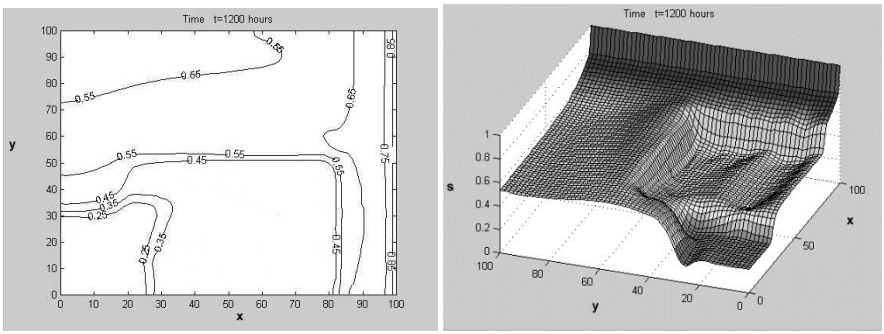

Fig.9. The water saturation at $t = 1200$ *hours.* $k_{D_1} = 0.20 \cdot 10^{-12}\ m^2$.

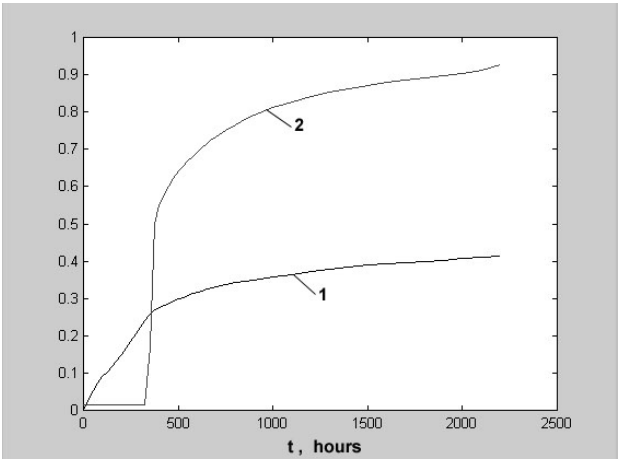

Fig.10. Characteristics of efficiency of oil recovery. Line 1 - $\theta(t)$ *, line 2 -* $\gamma(t)$*.*
$k_{D_1} = 0.20 \cdot 10^{-12}\ m^2$.

According to our results, we see that in the case considered in this paper, it is possible to develop only about 35 percents of oil by the usual water drive technology although 70 percents of oil is contained in the high penetration subregion. These results are in good correspondence with well-known practice.

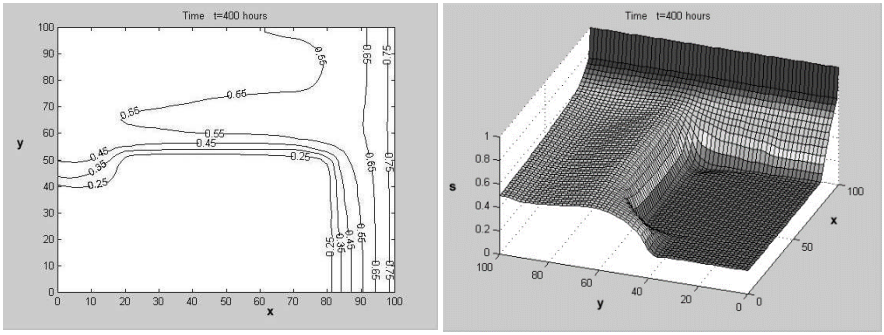

Fig.11. The water saturation at $t = 400$ *hours.* $k_{D_1} = 0.01 \cdot 10^{-12}\ m^2$.

5. Conclusions

Our high-precision numerical quasi-characteristics technique developed for the transport equation of hyperbolic allows us to obtain solutions of complicated porous media problem with essentially heterogeneous parameters without mesh fitting procedures on rough spatial meshes. This technique can be implemented even on small computers and workstations for fast evaluation and exact modeling of oil and gas development technological processes.

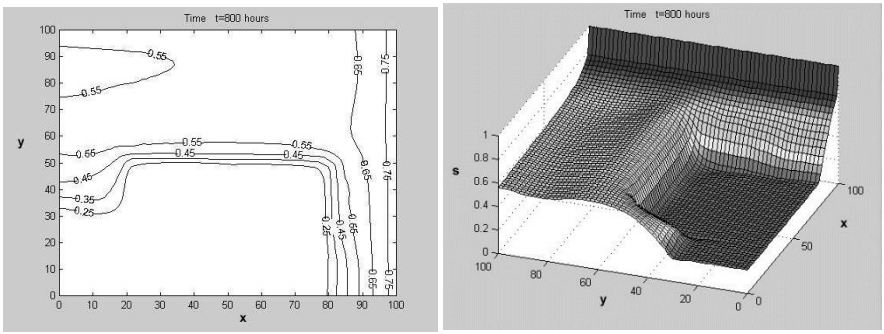

Fig.12. The water saturation at $t = 800$ *hours.* $k_{D_1} = 0.01 \cdot 10^{-12}\ m^2$.

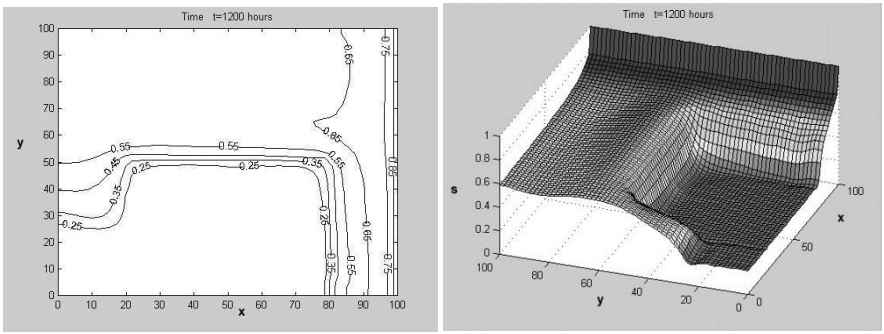

Fig.13. The water saturation at $t = 1200$ *hours.* $k_{D_1} = 0.01 \cdot 10^{-12}\ m^2$.

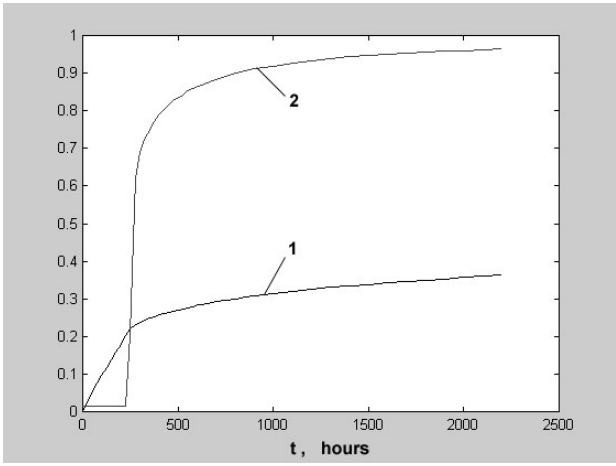

Fig.14. Characteristics of efficiency of oil recovery.
Line 1 - $\theta(t)$*, line 2 -* $\gamma(t)$*.* $k_{D_1} = 0.01 \cdot 10^{-12}\ m^2$.

References

[1]. B. Engquist, B. Sjogreen. High-Order Shock Capturing Methods. Computational Fluid Dynamics Review, John Willey and Sons, 1995, pp. 210–233.

[2]. E. Godlewsky, P.A. Raviart. Numerical Approximation of Hyperbolic Systems of Conservation Laws. Springer-Verlag, 1996, 524 p.

[3]. K.W. Morton. Numerical Solution of Convection-Diffusion Problems. Chapman and Hall, 1996, 384 p.

[4]. S.B. Hazra, P. Niyogi, S.K. Chakrabartty. Study in non-oscillatory schemes for shock computation using Euler equations. Computational Fluid Dynamics Journal, vol. 7, 1998, pp. 163–176.

[5]. Sh. Wo, B.M. Chen, J. Wang. A High-Order Godunov Method for One-Dimensional Convection-Diffusion-Reaction Problems. Numerical Methods for Partial Differential Equations, vol. 16, 2000, pp. 495–512.

[6]. M.P. Levin. A difference scheme of quasi-characteristics and its use to calculate supersonic gas flows. Computational Mathematics and Mathematical Physics, vol. 33, no. , 1993, pp. 113–121.

[7]. M.Yu. Zheltov, M.P. Levin. Application of the quasi-characteristics scheme for the two-phase flows through porous media. Computational Fluid Dynamics Journal, vol. 2, 1993, pp. 363–370.

[8]. M.P. Levin. Computation of 3-D supersonic flow with heat supply by explicit quasi-characteristics scheme. Computational Fluid Dynamics Journal, vol. 4, 1995, pp. 311–322.

[9]. M.P. Levin, L.V. Sidorov. Hybrid modification of the scheme of the method of quasi-characteristics on a pyramidal pattern. Computational Mathematics and Mathematical Physics, vol. 35, no. 2, 1995, pp. 253– 258.

[10]. M.P. Levin. Quasi-characteristics numerical schemes. In Hyperbolic Problems: Theory, Numerics, Application, Springer, 1999, pp. 619–628.

[11]. A.I. Ibragimov, M.P. Levin, L.V. Sidorov. Numerical investigation of two-phase fluid afflux to horizontal well by quasi-characteristics scheme. Computational Fluid Dynamics Journal, vol. 8, 2000, pp. 556–560.

[12]. V.M. Borisov, Yu.V. Kurilenko, I.E. Mikhailov, E.V. Nikolaevskaya. A method of characteristics for calculation of vortex spatial supersonic stationary flows. Computing Centre of USSR Academy of Sciences, Moscow, 1988 (in Russian).

[13]. E.V. Nikolaevskaya. One class of running finite difference schemes. Computing Centre of USSR Academy of Sciences, Moscow, 1987 (in Russian).

[14]. D.Y. Kwak, M.P. Levin. High-Resolution Monotone Schemes Based on Qasi-Characteristics Technique. Numerical Methods for Partial Differential Equations, vol. 17, 2001, 262–276

[15]. S.-H. Chou, Q. Li. Characteristics-Galerkin and mixed finite element approximation of contamination by compressible nuclear waste-disposal in porous media. Numerical Methods for Partial Differential Equations, vol. 12, 1996, pp. 315–332.

[16]. H. Wang, M. Al-Lawatia, A.S. Telyakovskiy. Runge-Kutta characteristic methods for first-order linear hyperbolic equations. Numerical Methods for Partial Differential Equations, vol. 13, 1997, pp. 617–661.

[17]. H. Wang, M. Al-Lawatia, R.C. Sharpley. A characteristic domain decomposition and space-time local refinement method for first-order linear hyperbolic equations with interfaces. Numerical Methods for Partial Differential Equations, vol. 15, 1999, pp. 1–28.

[18]. M. Marion, A. Mollard. A multilevel characteristics method for periodic convection-dominated diffusion problems. Numerical Methods for Partial Differential Equations, vol. 16, 2000, pp. 107–132.
[19]. C.N. Dawson, M.L. Martinez-Canales. A characteristic-Galerkin approximation to a system of shallow water equations. Numerische Mathematik, vol. 86, Issue 2, 2000, pp. 239–256.
[20]. Yu.P. Zheltov. Mechanics of Oil and Gas Bearing Formation. Nedra, Moscow, 1975 (in Russian).

Численное моделирование двухфазных течений через существенно гетерогенную пористую среду схемой квазихарактеристик высокого порядка

М.П. Левин <mlevin@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Рассматриваются вопросы численного моделирования нестационарных двухфазных потоков в пористых средах с существенно неоднородными свойствами с помощью численной схемы квазихарактеристик второго порядка аппроксимации. В отличие от известных схем высокого порядка, представленная схема имеет второй порядок аппроксимации в областях с большими градиентами решений, а также сохраняет монотонный характер решений. Это достигается за счет выбора итогового решения в каждой расчетной точке из нескольких допустимых решений с различными дисперсионными свойствами. Монотонный характер решения обеспечивается специальным критерием выбора решения, сформулированным в представленной работе. Этот критерий имеет локальный характер и удобен для параллельных вычислений. Эффективность подхода проиллюстрирована на решении задач вытеснения нефти водой в существенно неоднородных пористых пластах с коэффициентами абсолютной проницаемости, скачкообразно изменяющимися в десятки и сотни раз.

Ключевые слова: метод квазихарактеристик, параллельные вычисления, двухфазные течения, пористые гетерогенные среды.

DOI: 10.15514/ISPRAS-2018-30(5)-12

Для цитирования: Левин М.П. Численное моделирование двухфазных течений через существенно гетерогенную пористую среду схемой квазихарактеристик высокого порядка. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 197-212 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-12

Список литературы

[1]. B. Engquist, B. Sjogreen. High-Order Shock Capturing Methods. Computational Fluid Dynamics Review, John Willey and Sons, 1995, pp. 210–233.
[2]. E. Godlewsky, P.A. Raviart. Numerical Approximation of Hyperbolic Systems of Conservation Laws. Springer-Verlag, 1996, 524 p.
[3]. K.W. Morton. Numerical Solution of Convection-Diffusion Problems. Chapman and Hall, 1996, 384 p.

[4]. S.B. Hazra, P. Niyogi, S.K. Chakrabartty. Study in non-oscillatory schemes for shock computation using Euler equations. Computational Fluid Dynamics Journal, vol. 7, 1998, pp. 163–176.

[5]. Sh. Wo, B.M. Chen, J. Wang. A High-Order Godunov Method for One-Dimensional Convection-Diffusion-Reaction Problems. Numerical Methods for Partial Differential Equations, vol. 16, 2000, pp. 495–512.

[6]. М. П. Левин. Разностная схема квазихарактеристик и ее применение для расчета сверхзвуковых течений газа. Ж. вычисл. матем. и матем. физ., том 33, no. 1, 1993 г., стр. 131–141.

[7]. М. Ю. Желтов, М. П. Левин, Метод квазихарактеристик в задачах механики пористых сред. Ж. вычисл. матем. и матем. физ., том 33, no.10, 1993 г., стр. 1594–1599.

[8]. M.P. Levin. Computation of 3-D supersonic flow with heat supply by explicit quasi-characteristics scheme. Computational Fluid Dynamics Journal, vol. 4, 1995, pp. 311–322.

[9]. М. П. Левин, Л. В. Сидоров, Гибридная модификация схемы метода квазихарактеристик на пирамидальном шаблоне. Ж. вычисл. матем. и матем. физ., том 35, no.2, 1995 г., стр. 310–317.

[10]. M.P. Levin. Quasi-characteristics numerical schemes. In Hyperbolic Problems: Theory, Numerics, Application, Springer, 1999, pp. 619–628.

[11]. A.I. Ibragimov, M.P. Levin, L.V. Sidorov. Numerical investigation of two-phase fluid afflux to horizontal well by quasi-characteristics scheme. Computational Fluid Dynamics Journal, vol. 8, 2000, pp. 556–560.

[12]. В.М. Борисов, Ю.В. Куриленко, И.Е. Михайлов, Е.Л. Николаевская. Метод характеристик для расчета вихревых сверхзвуковых установившихся пространственных течений. М.: ВЦ АН СССР, 1988 г..

[13]. Е. Л. Николаевская. Об одном классе разностных схем бегущего счета. М.: ВЦ АН СССР, 1987 г..

[14]. D.Y. Kwak, M.P. Levin. High-Resolution Monotone Schemes Based on Qasi-Characteristics Technique. Numerical Methods for Partial Differential Equations, vol. 17, 2001, 262–276

[15]. S.-H. Chou, Q. Li. Characteristics-Galerkin and mixed finite element approximation of contamination by compressible nuclear waste-disposal in porous media. Numerical Methods for Partial Differential Equations, vol. 12, 1996, pp. 315–332.

[16]. H. Wang, M. Al-Lawatia, A.S. Telyakovskiy. Runge-Kutta characteristic methods for first-order linear hyperbolic equations. Numerical Methods for Partial Differential Equations, vol. 13, 1997, pp. 617–661.

[17]. H. Wang, M. Al-Lawatia, R.C. Sharpley. A characteristic domain decomposition and space-time local refinement method for first-order linear hyperbolic equations with interfaces. Numerical Methods for Partial Differential Equations, vol. 15, 1999, pp. 1–28.

[18]. M. Marion, A. Mollard. A multilevel characteristics method for periodic convection-dominated diffusion problems. Numerical Methods for Partial Differential Equations, vol. 16, 2000, pp. 107–132.

[19]. C.N. Dawson, M.L. Martinez-Canales. A characteristic-Galerkin approximation to a system of shallow water equations. Numerische Mathematik, vol. 86, Issue 2, 2000, pp. 239–256.

[20]. Ю.П. Желтов. Механика нефтегазоносного пласта. М.: Недра, 1975.

Specialized robust CFD RANS microscale meteorological model for modelling atmospheric processes and transport of contaminants in urban and industrial areas

O.S. Sorokovikova <olga_sorokov@mail.ru>
D.V. Dzama <diman_sw@mail.ru>
D.G. Asfandiyarov <dasfandiyarov@ibrae.ac.ru>
The Nuclear Safety Institute of the Russian Academy of Sciences,
52, Bolshaya Tulskaya Street, Moscow, 115191, Russia

Abstract. State-of-the-art models of dispersion of contamination in the urban environment and industrial areas employ a CFD approach in order to calculate turbulent characteristics of flow around buildings with complex geometry. The main area of application of these models is to facilitate licensing of potentially hazardous facilities and assessment of meteorological conditions in the urban environment. The usage of the most popular commercial CFD software with regard to modelling flows in the urban environment is significantly limited by the requirement for computational mesh refinement near the surface of the building in order to adequately resolve the characteristic scales in the viscous and buffer sublayers. On the other hand, models based on the traditional gaussian approach cannot take into account the complex aerodynamic effects in order to calculate turbulent characteristics of flow around buildings with complex geometry, including the subtleties concerning atmospheric emissions of gas-aerosol substances. Therefore, the authors developed a robust, highly specialized CFD-RANS model and a calculation code for modelling the atmospheric dispersion of contamination under conditions of a complex three-dimensional geometry that do not require mesh refinement. The authors verified this model using extensive database obtained both in the course of field experiments as well as of wind tunnel experiments. The verification results showed that the developed model satisfies the acceptance criteria for the quality of modelling along with foreign general-purpose codes and highly specialized codes.

Keywords: microscale meteorological models; passive tracer transport; dose calculation.

DOI: 10.15514/ISPRAS-2018-30(5)-13

For citation: Sorokovikova O.S., Dzama D.V., Asfandiyarov D.G. Specialized robust CFD RANS microscale meteorological model for modelling atmospheric processes and contamination transport in urban and industrial areas. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 213-234. DOI: 10.15514/ISPRAS-2018-30(5)-13

1. Introduction

A characteristic feature of the atmospheric transport of contaminants on a local scale is that the vertical and horizontal dimensions of the plume are comparable with the size of sharply outlined obstacles such as industrial buildings. Urban conditions entail three-dimensional flows around buildings. Hence, aerodynamic effects such as generation of recirculation zones, aerodynamic shadows, et cetera, dominate, thereby radically changing the local wind direction and velocity from the mean values at a larger scale. All these factors strongly influence the movement and dispersion of the contaminant plume. Figs. 1 and 2 show typical examples of the volume concentration distribution of contaminants near the surface obtained by three-dimensional modeling under urban conditions and under assumption of a smooth surface without obstacles with a uniform vertical profile of the wind velocity. A transition from red to blue signifies a difference in concentration of several orders of magnitude.

As can be seen from the figures, simplified approaches that do not take into account the real geometry of the obstacles lead to a non-realistic Gaussian distribution of contaminants.

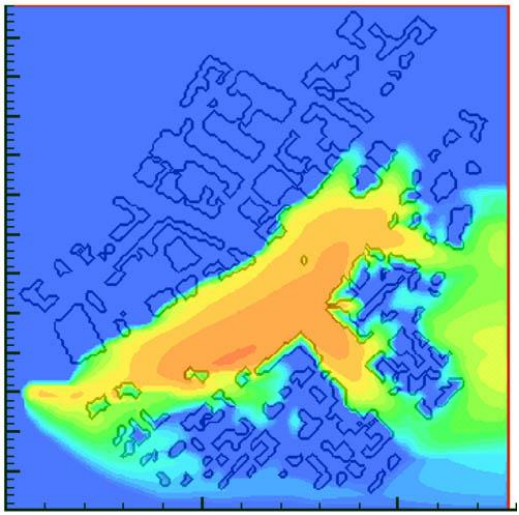

Fig. 1. Pollutant concentration distribution taking into account the 3D model of the urban area

State-of-the-art models of atmospheric dispersion of pollutants under urban conditions use the CFD approach to calculate the characteristics of the turbulent flow around buildings of arbitrary geometry. In accordance with the international classification, such models belong to the class of microscale meteorological models (MMM) [1]. The main area of application of such models is the analysis of the

environmental impact of industrial facilities, as well as the assessment of meteorological conditions in urban areas.

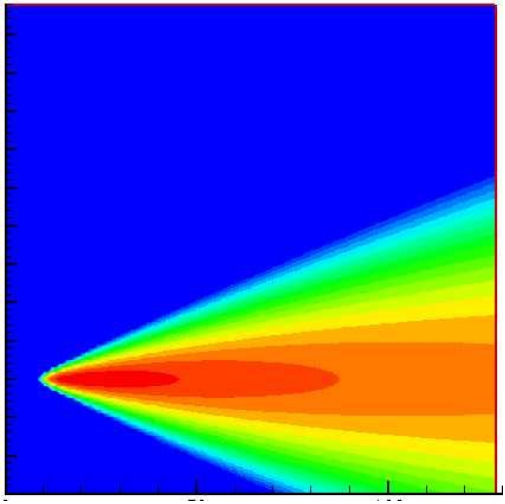

Fig. 2. Pollutant concentration distribution with a uniform wind field without buildings

A distinctive feature of the modern approach to modeling the transport of contaminants near industrial facilities and urban buildings is that it takes into account the three-dimensional pattern of flow around buildings. As a result, it has become possible to model aerodynamic effects and to obtain more realistic concentration distributions, which is fundamentally impossible within the framework of traditional Gaussian techniques.

Popular commercial CFD codes for pollutants atmospheric dispersion that take into account realistic geometry of urban or industrial buildings are mostly limited to simplified problems. The use of such codes for modeling flows around buildings with complex geometry is limited by the need to refine the computational mesh near the surfaces of buildings and the ground. Mesh refinement is necessary for the correct modelling of processes in the viscous and buffer layers. Though, such a significant mesh refinement can be very computationally demanding.

Current trends in tackling this problem are to employ highly specialized microscale meteorological models designed to simulate atmospheric processes taking into account 3D geometry of buildings.

A distinctive feature of highly specialized models is the parametrization of turbulent flow near the surfaces of the ground and buildings based on the Monin-Obukhov theory. Thus, one can avoid mesh refinement to resolve the viscous and buffer layers, thereby alleviating computational demands, in contrast with general-purpose software. A comparison was made with several foreign models of this class put to practical use.

This paper presents a model related to the MMM class. In comparison with other models of this class, its distinctive feature is the parametrization of the turbulent heat flux and the kinetic energy of turbulence near the surface under both stable and unstable temperature stratification.

In addition to calculating the concentration fields, the developed code calculates the doses of external and internal exposure (from inhalation). External exposure is divided into radiation from the cloud and the surface. Calculation of radiation from the cloud and the surface takes into account the shielding effect assuming that large buildings represented in a three-dimensional model of an industrial object completely absorb radiation.

Therefore, the developed model and the code allows modelling flows around buildings and calculation of volume and surface concentrations as well as the radiation situation in the territory of an industrial facility or a city. However, this paper describes only the MMM and verification of a part of the general software complex modelling key parameters of atmospheric non-isotropic processes (wind, turbulence). These parameters are the input data for modelling pollutants transport in case of complex geometry.

There is an analog put to practical use in Europe - a commercial software MISKAM (passive tracer, chemical compounds in urban areas, no version dealing with radioactivity issues) [2].

In the US, there is the FEM3MP model of the Livermore laboratory. It takes into account the specifics of radioactive contamination, but the model is not available to third-party users [3].

2. Model Description

The paper presents a CFD-RANS model belonging to the category of microscale meteorological models. This model allows obtaining fields of contaminant concentrations in the atmosphere, surface deposition of buildings and the ground as a result of the gas-aerosol release, taking into account the real geometry of the object, the stratification of the atmosphere, and heterogeneous turbulence in the atmospheric boundary layer.

The model is based on the incompressible Reynolds-averaged Navier-Stokes equations. Instead of using the wall functions in the first computational cell, the parametrization of heat flux and impulse is used in accordance with [4].

Input vertical profiles of speed and temperature are constructed in accordance with the model of the atmospheric boundary layer using the classification of the atmospheric stability classes according to Turner.

All calculated variables are treated as deviations from the hydrostatic balance. The Boussinesq approximation is used.

All physical quantities in the model are dimensional, the SI system is used.

The basic equations are as follows:

- Continuity equation

$$\mathrm{div}\,\vec{u} = 0 \qquad (1)$$

- Momentum transfer equation

$$\rho_0 \frac{\partial \vec{u}}{\partial t} + \rho_0 \left(\vec{u}\vec{\nabla}\right)\vec{u} = -\vec{\nabla}\delta P + \vec{\nabla}\left(\rho_0 \nu_T \vec{\nabla}\right)\vec{u} + \rho_0 \vec{g}\frac{\delta\theta}{\theta_0} + \vec{f} \qquad (2)$$

- Heat transfer equation

$$\frac{d\theta}{dt} = \frac{\partial\theta}{\partial t} + \vec{u}\vec{\nabla}\theta = \vec{\nabla}\left(\chi_T \vec{\nabla}\theta\right) \qquad (3)$$

- The passive contaminant (*i*-th radionuclide) transfer equation

$$\frac{\partial C_i}{\partial t} + (\vec{u} + \vec{w}_i)\vec{\nabla}C_i = \vec{\nabla}\left(D_T \vec{\nabla} C_i\right) + Q_{C_i} \qquad (4)$$

In (1) – (4), $\vec{u}$ – three-dimensional vector of averaged flow velocity (m/s); ρ_0 – undisturbed input air density near the surface of the earth (constant, ~ 1 kg/m^3); t – time (s); δP – pressure deviation from unperturbed hydrostatic at a given height with neutral stratification (Pa); ν_T – model coefficient of turbulent viscosity (m^2/s); $\vec{g}$ – the vector of gravitational acceleration (m/s^2); θ_0 – unperturbed value of potential temperature at a given height (K); $\delta\theta$ – the deviation of the potential temperature from the unperturbed value at a given height (K); $\vec{f}$ – possible force effect per unit volume of air (N /m^3); χ_T – model coefficient of turbulent thermal diffusivity associated with constant factor ν_T; C_i – concentration value of the *i*-th component of the impurity (kg/m^3 or Bq/m^3 for radioactive impurity); $\vec{w}_i$ – velocity vector of gravitational subsidence (for the aerosol component of the impurity, m/s); D_T – model coefficient of turbulent diffusion of the contaminants associated with constant factor ν_T (m^2/s); Q_{Ci} – dependent on the coordinate and time of the emission power of the *i*-th component (kg/s or Bq/s).

A two-layer model of turbulence is used for the closure of the basic equations.

The classical model of turbulence $k-\varepsilon$ is applied to non-surface cells, that is, to those that do not have solid faces.

The kinetic energy equation for turbulence *k* is as follows:

$$\frac{\partial k}{\partial t} + \vec{\nabla}\left(k\vec{u}\right) = \vec{\nabla}\left(\frac{\nu_T}{\sigma_k}\vec{\nabla}k\right) + S + G - \varepsilon, \qquad (5)$$

where *k* is the kinetic energy of turbulence (m^2/s^2); σ_k – dimensionless empirical constant (equal to 1). The term *G* on the right-hand side of the equation is responsible for the generation of turbulent energy due to temperature stratification. The parameter *G* is proportional to the gradient of potential temperature:

$$\frac{\vec{g}}{\theta}\left(\overline{\vec{u}'\theta'}\right) = G = \frac{\nu_T}{\sigma_\theta}\frac{\vec{g}}{\theta}\vec{\nabla}\bar{\theta} \quad (6)$$

In (6), $\vec{u}'$ and θ' – pulsations of the vector of the flow velocity and potential temperature, respectively; σ_θ – dimensionless empirical constant (0.9).

The first term on the right-hand side of the equation (5) is a parameterization of the transfer of turbulent energy by velocity pulsations:

$$-\overline{\frac{\partial u_j' k'}{\partial x_j}} = \frac{\partial}{\partial x_j}\left(\frac{\nu_T}{\sigma_k}\frac{\partial k}{\partial x_j}\right) \quad (7)$$

The parameter S on the right side of the equation (5) is responsible for the generation of turbulence energy due to shear deformations and is determined as follows (summation is performed over repeated indices in accordance with the Einstein rule):

$$S = \nu_T\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right]\frac{\partial u_i}{\partial x_j} \quad (8)$$

In the definition (8), u_i is the i-th component of the averaged flow velocity. The transfer equation for turbulent energy dissipation is as follows:

$$\frac{\partial \varepsilon}{\partial t} + \vec{\nabla}\left(\varepsilon\,\vec{\mathrm{u}}\right) = \vec{\nabla}\left(\frac{\mu_T}{\sigma_\varepsilon}\vec{\nabla}\varepsilon\right) + \frac{\varepsilon}{k}\left(C_{\varepsilon 1}S + C_{\varepsilon 3}G\right)S - C_{\varepsilon 2}\frac{\varepsilon^2}{k} \quad (9)$$

In the equation (9), ε – the turbulent energy dissipation rate (m^2/s^3); σ_ε, $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, and $C_{\varepsilon 3}$ are dimensionless empirical constants, which are equal to, respectively: 1.3; 1.21; 1.92; 1.44 (for atmospheric problems).

The transfer equations (5) and (9) have five empirical constants: $C_{\varepsilon 1}$, $C_{\varepsilon 2}$, $C_{\varepsilon 3}$, σ_k, σ_ε. In this form, model equations $k-\varepsilon$ are widely used to simulate turbulence, as, for example, in [5].

The coefficient of turbulent viscosity, which is used in the Reynolds-averaged Navier-Stokes equations, is calculated using the following formula:

$$\nu_T = C_\mu\frac{k^2}{\varepsilon} \quad (10)$$

In (10), C_μ is a dimensionless empirical constant (0.03 – for atmospheric problems). In contrast with the classical k–ε turbulence model, the following parametrization for k and ε is used in the subsurface cells. These values are determined by the value of the dynamic velocity (friction velocity), which is determined by the value of the tangential component of the velocity along a solid surface and also depends on the stability class of the atmosphere. For example, the following relation is used for unstable (A, B, C according to Turner's classification

with minor modifications) and neutral stratification (D according to the same classification) [4]:

$$u(z)=\frac{u^*}{\kappa}\left[\ln\left(\frac{z}{r}\right)-\left\{\ln\left(\frac{1+\xi^2}{2}\right)+2\ln\left(\frac{1+\xi}{2}\right)-2\mathrm{arctg}(\xi)+\frac{\pi}{2}\right\}\right], \quad (11)$$

where:

- z – the distance from the measurement point of the tangential component of the flow velocity to the streamlined surface (m);
- u(z) – the tangential component of the averaged flow velocity (m/s);
- u* – the actual value of the dynamic velocity or friction velocity (m/s);
- κ=0.41 – the Karman constant;
- r – the surface roughness, m;
- γ – a dimensionless constant, equal to 15;
- L – the Monin-Obukhov scale, which depends on the stability class of the atmosphere [6] (the values are given in Table 1);
- $\xi=(1-\gamma z/L)^{1/4}$, dimensionless parameter.

Table 1. Monin-Obukhov scale depending on the stability class

Atmosphere stability class	A	B	C	D	E	F	G
Mean value	−5	−25	−70	−500	55	5	1
Minimum value	−10	−40	−100	−∞	10	1	0
Maximum value	0	−10	−40	±100	100	10	1

To determine the turbulent heat flux $\overline{w'\theta'}$, where w' – the pulsation of the vertical component of the flow velocity (m/s), and θ' – the potential temperature pulsation (K), (12) and (13) parameterizations are used, which are given below [7]. In the case of unstable and neutral stratification, these parametrizations are as follows:

$$\overline{w'\theta'}=-u^*\frac{\theta^{AIR}-\theta^{GROUND}}{\frac{R}{\kappa}\left(\ln\left(\frac{z}{r}\right)-2\ln\left(\frac{1+\eta^2}{2}\right)\right)} \quad (12)$$

For stable stratification:

$$\overline{w'\theta'}=-u^*\frac{\theta^{AIR}-\theta^{GROUND}}{\frac{R}{\kappa}\left(\ln\left(\frac{z}{r}\right)+\frac{\beta z}{RL}\right)} \quad (13)$$

where $\eta = \left(1 - \lambda z / L\right)^{1/4}$, R=0.74, λ=9; β=4.7 (all are dimensionless constants); θ^{AIR} – the potential temperature in the surface layer; θ^{GROUND} – the potential temperature of the earth, which is kept constant during calculations.

The use of such parametrizations limits the size of the computational mesh to be 20–30 times larger than the roughness value.

To calculate k and ε, (14) and (15) formulas are used in the near-surface cells, accordingly:

$$k = \frac{u_*^2}{C_\mu^{1/2}}, \quad (14)$$

$$\varepsilon = \frac{u_*^3}{\kappa z}. \quad (15)$$

The model requires the following input data: three-dimensional Cartesian grid; three-dimensional models of the buildings; wind speed and direction at a height of 10 m; atmospheric stability class; geographical latitude of the object; roughness value of the underlying surface. The initial and boundary conditions for the hydrothermodynamic problem are set automatically from the data listed above, which is provided by the user.

To solve the transfer problem, it is necessary to set the parameters of the source of release: the release rate, the nuclide composition of the source, the position of the source in space. For each nuclide, the dry precipitation rate is required. If necessary, there may be several point sources, and they can be placed at different points in the computational domain.

3. Verification and Validation

Below there are some results of verification of the model and examples of its application.

Studies have shown that the existing data sets obtained during the experiments are not always suitable for the verification of microscale models. This impelled the international scientific community to initiate the creation of a database that would be more suitable for these purposes [8].

There are two approaches to solve this problem. The first approach is to create a database for verification by collecting information from experiments conducted in wind tunnels. The second approach gives priority to experiments conducted under natural atmospheric conditions [9].

Both approaches have their pros and cons, it is very difficult to simulate some phenomena in a wind tunnel experiment, such as: stratification, thermal effects as a result of heating or cooling of building surfaces, chemical reactions, and aerosol precipitation.

On the contrary, a coarse measurement grid, which is typical of field experiments, does not allow accurate estimation of the parameters of the inhomogeneous and unsteady flow, which is necessary for comparison with the data obtained by modeling.

At the present time, there exist quantitative parameters of the quality of simulation results obtained by using microscale meteorological models. In the literature, the simulation results are mainly characterized by two quantitative parameters.

Quantitative assessment of simulations is based on the comparison of two series of the same size consisting of C_{calc} and C_{obs} – model (calculated) and measured values for a given physical quantity, respectively. Each element of these two series corresponds to a certain point of measurement of the physical quantity value, which can be either one of the components of the flow velocity or tracer concentration.

In order to verify the model, the quality of modeling was estimated by the following values [1]:

- FA-2 (factor of 2 of observation) shows the proportion of the total number of measurement points for which the condition (17) is met:

$$FA-2=\frac{N}{n}=\frac{\sum_{i=1}^{n}N_i}{n}, \quad (16)$$

$$N_i=\begin{cases}1, \text{ if } 1/2\le C^i_{calc}/C^i_{obs}\le 2\\ 1, \text{ if } C^i_{obs}\le W\\ 0, \text{ else}\end{cases}. \quad (17)$$

- Hit Rate is defined as:

$$HR=\frac{N}{n}=\frac{1}{n}\sum_{i=1}^{n}N_i, \quad (18)$$

$$N_i=\begin{cases}1, \text{ if } \left|\frac{C^i_{calc}-C^i_{obs}}{C^i_{obs}}\right|\le D\\ 1, \text{ if } \left|C^i_{calc}-C^i_{obs}\right|\le W\\ 0, \text{ else}\end{cases}. \quad (19)$$

In (17) and (19), C^i_{calc} and C^i_{obs} – elements of the series of the calculated and measured physical quantities with the same index. In (16), N – the total number of pairs (measurement points) for which the condition (17) is fulfilled, n – dimensions of the input arrays (measured and calculated), that is, the total number of

measurement points. In (17), W – the threshold value of the measured flow velocity or concentration, below which the condition (17) is considered to be satisfied regardless of the value of the calculated flow velocity or concentration.

Similarly, in the ratio (18), N – the total number of pairs for which the condition (19) is fulfilled; D – the accepted relative error of calculation of a physical quantity; W – the threshold value of the absolute error of calculation of the physical parameter, below which the condition (19) is considered to be satisfied regardless of the actual measured and calculated values.

The parameter D takes into account the relative uncertainty of the comparison, and the parameter W reflects the measurement uncertainty in the experiment. The value of D recommended by the expert community is 25% [5, 10]. The value of W is determined using statistical analysis of the variation of measurements in a series of experiments conducted under the same conditions.

Within the COST732 project [10], a database called CEDVAL [1] (compilation of experimental data for the validation of microscale meteorological models) was created, consisting of a set of experiments in a wind tunnel. The geometry of the obstacles was of varying difficulty: from one obstacle in the form of a rectangular parallelepiped to experiments in which there were four obstacles with slanted roofs, as well as experiments with an almost regular arrangement of 21 a rectangular parallelepipeds.

Let us focus on the experiments themselves. Verification was performed for a series of experiments A and B.

Experiments of the A1-1 series were characterized by a frontal flow around one obstacle in the form of a rectangular parallelepiped, the dimensions of which are 20 m × 30 m × 50 m. In this series of experiments, pairs of flow velocity components in (u, w) and (u, v) were measured in two mutually perpendicular planes.

In experiments of series A1-2 and A1-3, a rectangular parallelepiped strongly elongated along the Y-axis, the dimensions of which are 25 m × 324 m × 25 m, appeared as an obstacle. Wind direction is along the X-axis. In experiments A1-2, measurements were taken in the vertical plane running through the center of the obstacle. The pair of components of the flow velocity (u, w) were measured. In experiments A1-3, measurements were carried out both in the horizontal and in the vertical plane. In the horizontal plane, the components (u, v) were measured, and the vertical components (u, w).

In a series of experiments A1-4, a cube measuring 25 m was frontally flown around. The measurements of the components of the flow velocity were carried out in several horizontal and vertical sections. In horizontal sections, pairs of components (u, v) were measured, in vertical sections – pairs (u, w).

In experiments of the A1-5 series, a rectangular parallelepiped measuring 20 m × 30 m × 25 m was frontally flown around. A constant point source of the tracer acted

near the obstacle. In the experiments, the steady-state values of the tracer concentration in several planes of different orientations were measured.

In experiments of the A1-6 series, a cube measuring 25 meters flowed around one of the faces at an angle of 45°. In this series of experiments, pairs of flow velocity components (u, v) and (u, w) were measured in the horizontal and vertical planes, respectively.

In the experiments of A1-7 series, a cubic obstacle was also considered, but the flow was made at an angle of 40°. In this series of experiments, several components of the flow velocity were measured in several planes of different orientations.

In the experiments of B1-1 series, there were 20 obstacles measuring 20 m × 30 m × 25 m each, arranged in a 7 × 3 pattern (one building of 21 points was missing). Measurements of a pair of velocities (u, v) were carried out in one horizontal plane, a pair (u, w) – in four vertical planes. In addition, a constant point source of the tracer was present in the experiments and the steady-state field of its concentration in one plane was measured.

In the experiments of B1-2 series, there were 4 ring obstacles with dimensions of 250 m × 250 m × 60 m, arranged according to a 2 × 2 scheme. An annular obstacle could be obtained geometrically by separating a smaller parallelepiped from a rectangular parallelepiped with dimensions of 130 m × 130 m × 60 m. The pairs of velocity components (u, v) and (u, w) were measured in two horizontal and two vertical planes.

In the experiments of B1-3 series, 4 obstacles were distant along the X-axis to a greater distance than in the experiments of B1-2 series. The pairs of components (u, w) and (u, v) were measured in the vertical and horizontal planes.

In the experiments of B1-4, B1-5, and B1-6 series, there were similar ring obstacles, but with a more complex roof shape, part of which had a sloping shape. The wind direction in these experiments differed. In all experiments, pairs of components of the flow velocity (u, w) and (u, v) were measured in several planes of different orientations.

The verification results based on the CEDVAL database include about 20,000 single measurements and are presented in Table 2. This table presents the parameters FA-2 and Hit Rate for the three components of the flow velocity.

The average value of FA-2 and Hit Rate is defined as the weighted sum of these characteristics over all experiments. The weight in each experiment is equal to the ratio of the number of data points in this experiment to the total number of measurement points in all experiments.

Table 2. The total values of the statistical parameters FA-2 and HIT RATE for the three components of the flow velocity for all experiments (series A and B)

Criterion	Value
FA-2 (U)	87%

FA-2 (V)	96%
FA-2 (W)	93%
Hit-rate (U)	76%
Hit-rate (V)	82%
Hit-rate (W)	75%

In Table 2, U, V, and W are the components of the flow velocity along the X, Y, and Z axes, respectively. The verification results in Table 2 are given for all experiments of A and B series. These results were obtained by combining the data sets in all experiments. For several series of experiments, the verification results are given below.

In accordance with the COST732 project documents, the acceptance criterion for microscale meteorological models is 66% and 55% for FA-2 and Hit Rate, respectively.

Methods for analyzing the results of modeling against measurement data have been fairly well developed and are widely used in practice in relation to microscale meteorological models. For verification other quantitative characteristics are also used.

PCC (Pearson correlation coefficient):

$$P = \frac{\sum_{i=1}^{n}\left(C_{calc}^{i} - \bar{C}_{calc}\right)\left(C_{obs}^{i} - \bar{C}_{obs}\right)}{\sqrt{\sum_{i=1}^{n}\left(C_{calc}^{i} - \bar{C}_{calc}\right)^{2}\sum_{i=1}^{n}\left(C_{obs}^{i} - \bar{C}_{obs}\right)^{2}}}, \quad (20)$$

where P is in the range from –1 to 1. The ideal case for the model is achieved at P = 1. In the ratio (20), $\bar{C}_{calc}$ and $\bar{C}_{obs}$ are the average values of the calculated and measured physical quantity over the whole array. Each of the three summations is performed over all pairs of arrays of measured and calculated values.

Statistical parameters formulated in terms of overestimation or underestimation of measured values and representing the average offset of the calculated values are also widely used. The following criteria are the most popular in this category.

BIAS estimates the deviation of the calculated average values from the measured, expressing the deviation in physical units of measurement (in this case, in units of concentration):

$$BIAS = \bar{C}_{calc} - \bar{C}_{obs} = \frac{1}{N}\sum_{i=1}^{N}\left(C_{calc}^{i} - C_{obs}^{i}\right) \quad (21)$$

The BIAS value shows the absolute value of underestimating or overestimating the calculated values in comparison with the measured ones.

SAA (Scaled Average Angle difference) is a weighted average angular deviation, which is used less frequently [11], but was used to verify the developed model:

$$SAA = \frac{\sum_{i=1}^{N} |U_i||\varphi_i|}{\sum_{i=1}^{N} |U_i|} \quad (22)$$

where φ_i – the angle between the calculated and measured flow velocity, $|U_i|$ – the module of the measured flow velocity at the *i*-th point.

SAA is used if there is a detailed measurement network that can contain data on the direction and magnitude of the wind in several sections. It characterizes the accuracy of wind direction modeling by calculating a weighted sum with weights equal to the modulus of the flow velocity. The greater the flow velocity at a given point, the more it contributes to the total amount.

The results of the statistical comparison of calculated and measured values for the A1 experiment are shown below (in Tables 3-5).

Table 3. Statistical criteria for the longitudinal velocity component (along the main flow). Vertical section. Experiment A1

Criterion	Value	Best value
FA-2	98%	100%
HR	90%	100%
BIAS	–0.09260	0
PCC	0.96194	1
SAA	6.85260	0

Table 4. Statistical criteria for the vertical velocity component. Vertical section. Experiment A1

Criterion	Value	Best value
FA-2	98%	100%
HR	90%	100%
BIAS	–0.09260	0
PCC	0.96194	1
SAA	6.85260	0

Table 5. Statistical criteria for the longitudinal velocity component (along the main flow). Horizontal section. Experiment A1

Criterion	Value	Best value

FA–2	93%	100%
HR	75%	100%
BIAS	–0.46255	0
PCC	0.93649	1
SAA	7.61284	0

In addition to laboratory experiments, the comparison of simulated and measured concentration values was carried out using the data of the experiment JOINT URBAN 2003 (downtown Oklahoma). The essence of the experiments consisted in the artificial creation of a permanent source of atmospheric emission of a passive tracer for the subsequent restoration of the pollution pattern under the conditions of the complex urban development of the center of Oklahoma. Elegas (SF_6) was used as a passive tracer that does not enter into chemical reactions and does not precipitate on solid surfaces. In one of the experiments, the results of which were used to compare the results of the described model, SF6 gas was ejected at a constant intensity for half an hour with neutral temperature stratification. During the same period of time, the concentration of SF6 gas in the air was continuously monitored with high-precision equipment, both using stationary and mobile laboratories (installed on vehicles). The steady-state concentrations of SF6-gas obtained by the described model with the highest readings of high-precision sensors were compared. In the same period of time, the flow was measured at various altitudes with the help of high-precision equipment. This dataset allowed finding an approximation of the vertical input profile of the horizontal component of the wind speed, which was used to simulate hydrodynamics using the developed model.

Fig. 3 shows the position of the source against the background of the three-dimensional model of the city. Fig. 4 shows the concentration isosurfaces. Fig. 5 shows the position of the source against the background of the map and the positions of the concentration measurement points. Fig. 6 shows profiles of measured and calculated tracer concentrations depending on the number of the measurement point. Green squares represent the calculation results for the developed model, blue triangles – measurement data by stationary stations.

The tracer concentration field on the streets of a real city can be very complex. The ratio of concentrations at close points can be at the level of three orders of magnitude. A significant discrepancy was obtained at station number 13, where the model results are underestimated compared to the measurement results. However, the absolute maximum concentration (measurement station 20) corresponds to the model results with a relative accuracy of 5%.

Let us consider the situation associated with the measuring station 13, where there was the greatest discrepancy. Fig. 7 shows the location of this station (and station 14), as well as the isolines of the calculated tracer concentration in this area. It can be noted that the concentration field of the tracer in this area is highly non-uniform

– the horizontal gradient is high. Consequently, the concentration of the tracer in the vicinity of point 13 is much greater than in the point itself.

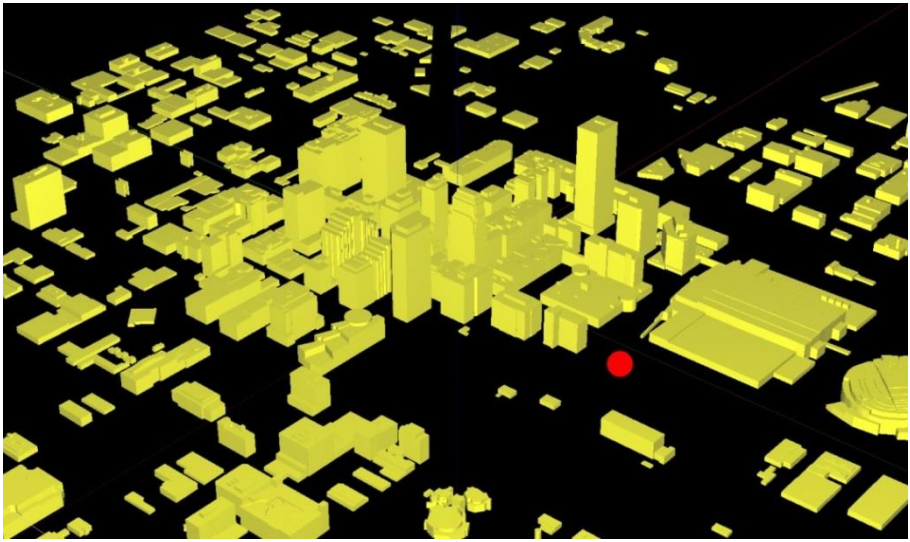

Fig. 3. 3D model of the center of Oklahoma city

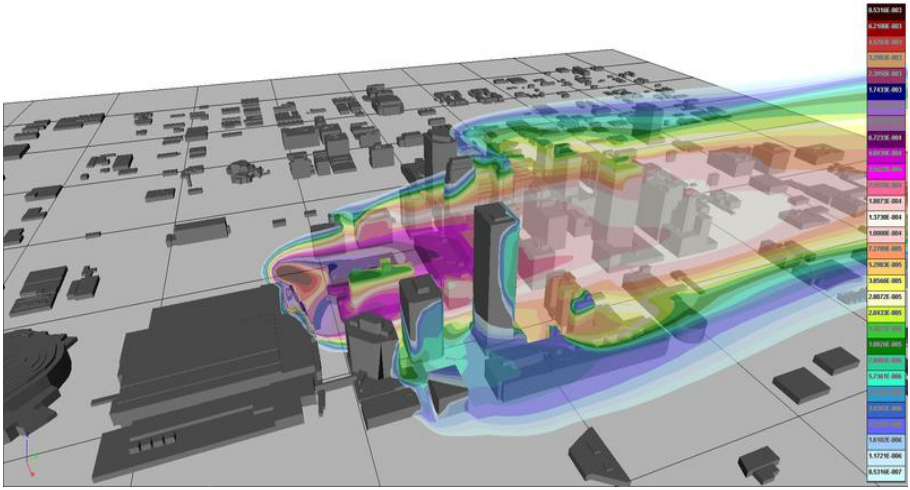

Fig. 4. Isosurfaces of concentration (results of modeling)

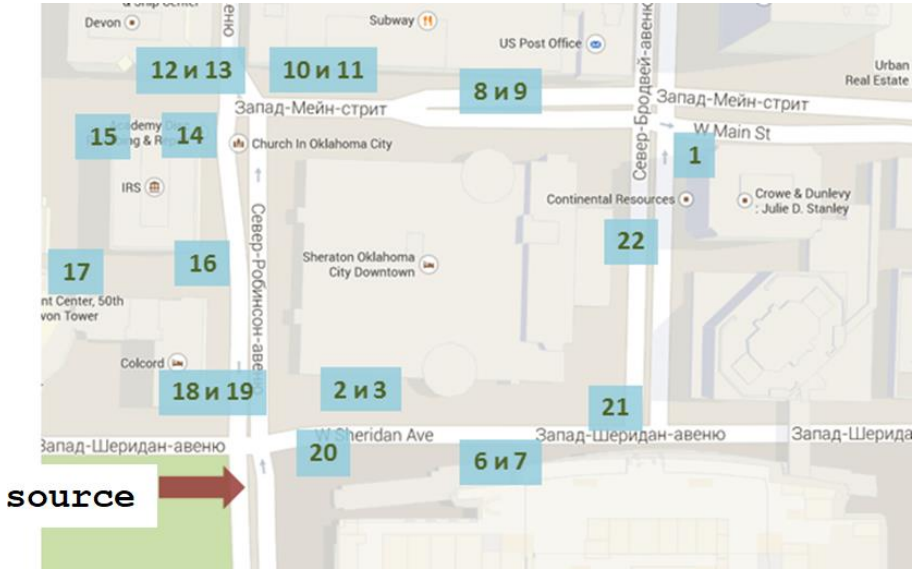

Fig. 5. Release point and measurement stations

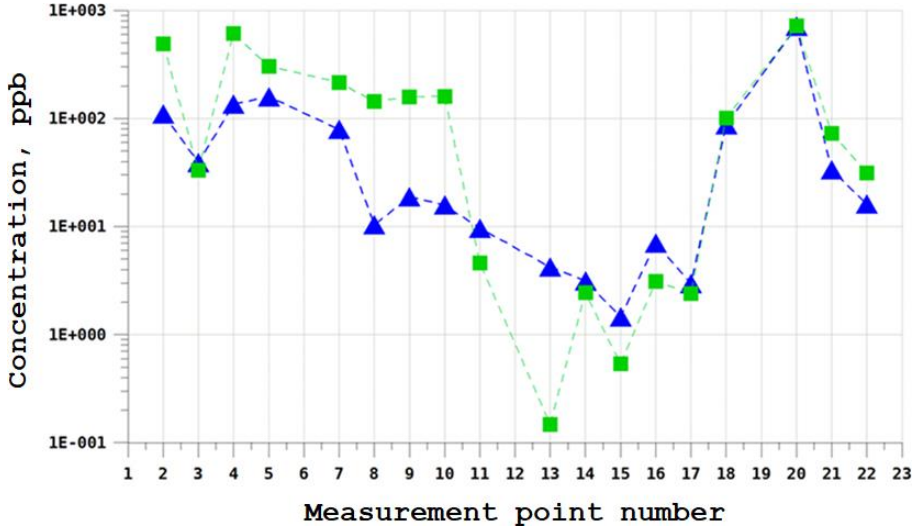

Fig. 6. Model results against experimental values

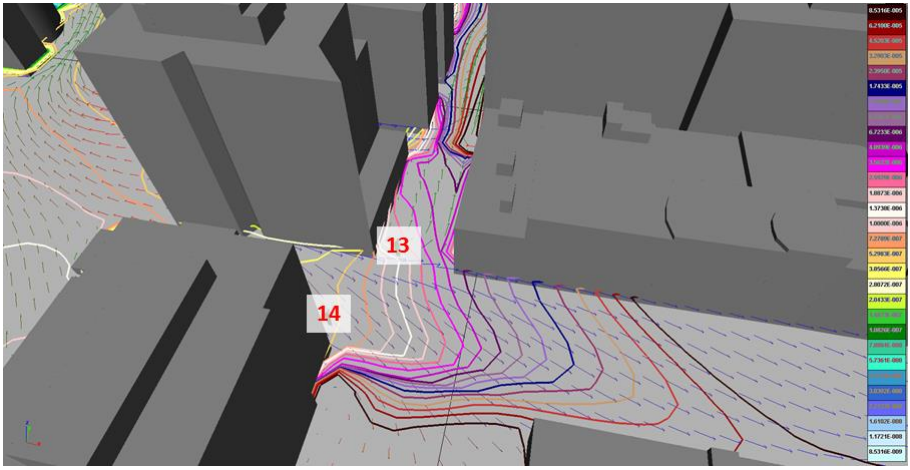

Fig. 7. Isolines of surface concentration in the vicinity of station 13 and 14

In addition to the calculations of the concentration fields in the model, the doses of external and internal exposure (from inhalation) are calculated. External exposure is divided into radiation from the cloud and the contaminated surface. Calculation of doses from external exposure is made taking into account the effect of radiation shielding by buildings, on the basis of the assumption that large buildings represented in a three-dimensional model of an industrial object completely absorb the dose-forming radiation.

The results of simulation at the Beloyarsk NPP is shown in Fig. 8 as an example of practical usage of the model. A hypothetical scenario characterized by loss of systemic and reliable power supply (failure of active reactor shutdown systems, failure of the EHRS (emergency heat removal system)), depressurization of 25% of nuclear reactor core fuel elements, which have a maximum burnout was considered. One of the nuclides of the release is ^{137}Cs ($5 \cdot 10^{13}$ Bq). The duration of the release was 30 minutes and the source was located at the level of the roof of the reactor building (in Fig. 8 – building with a pipe). Fig. 8 shows distribution of ^{137}Cs surface contamination of land and buildings (Bq/m^2) at the time of the end of the source action.

4. Cross-verification of the developed code with foreign calculation codes

The results of the cross-verification based on the Hit Rate statistical parameter for the vertical component of the flow velocity of the developed model with different models are shown in Fig. 9. The measurement data was obtained as a result of a tunnel experiment conducted by the University of Hamburg in the framework of the COST732 project.

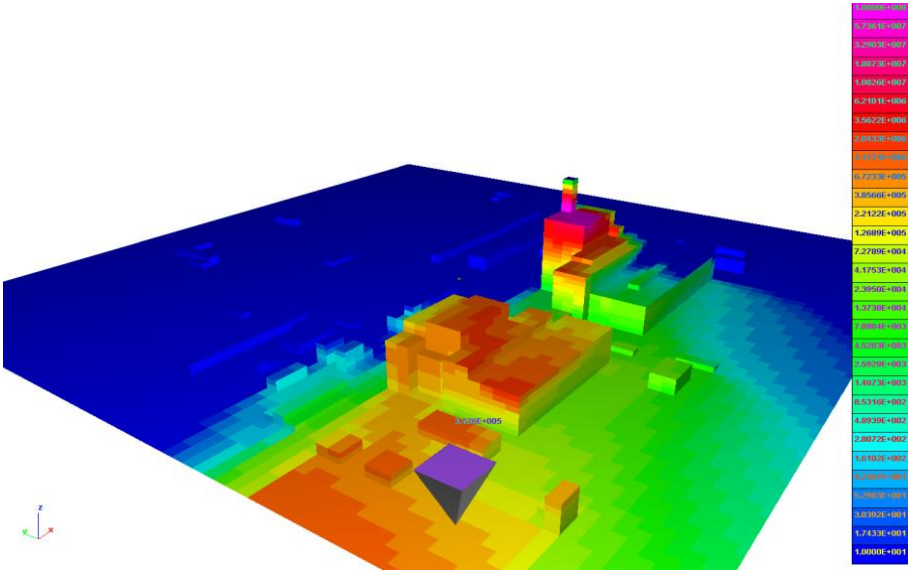

Fig. 8. Visualization of a concentration deposition field in the area of the Beloyarsk NPP.

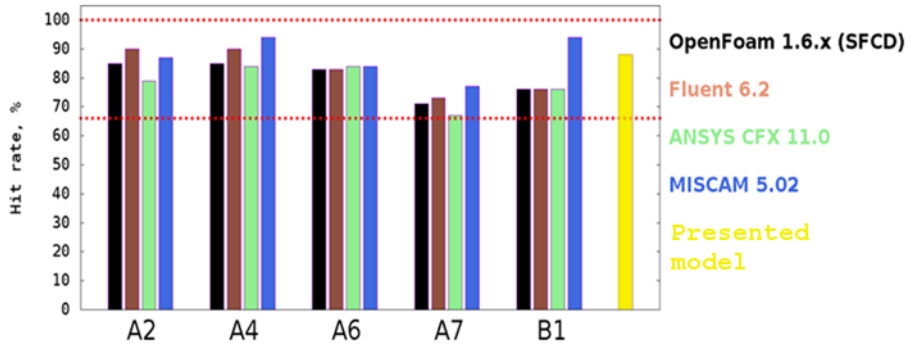

Fig. 9. Results of cross-verification of the developed model

Cross-verification was carried out using the results of modeling of five different experiments (A2, A4, A6, A7, B1 of the COST 732 database – Fig. 9) by four different calculation codes. Each component of the flow velocity was compared. Fig. 9 shows the results of verification of the longitudinal (along the X-axis — typical wind direction in all experiments) components of the flow velocity. The Hit Rate values gained by foreign models are presented for several experiments of series A and one experiment of series B separately. For the model described in this

paper, the Hit Rate value is given in sum for all the experiments carried out, both series A and series B.

Thus, the verification of the model on an international database showed that the calculation code and the model meet the criterion of the modeling quality developed by the international expert community [9].

5. Conclusion

In accordance with current trends in applied computational meteorology, a reliable CFD model has been developed. The verification matrix of the developed model and design code contains various data obtained from both large-scale field experiments in a real city (Oklahoma City) and from laboratory experiments. It is shown that the developed model meets the quality criteria of the simulation, defined by the expert community for models of the same class.

References

[1]. Michael Schatzmann, Helge Olesen, Jörg Franke: COST 732 Model Evaluation Case Studies: Approach and Results. University of Hamburg Meteorological Institute Centre for Marine and Atmospheric Sciences, 2010, http://www.mi.uni-hamburg.de/Official-Documents.5849.0.html

[2]. SoundPlan. MISKAM advanced. http://www.soundplan.eu/english/soundplan-air-pollution/miskam-advanced/

[3]. Stevens T. Chan, Martin J. Leach. A Validation of FEM3MP with Joint Urban 2003 Data. Journal of applied meteorology and climatology. Vol. 46. 2007. pp. 2127-2146 Data base CEDVAL at Hamburg University, http://www.mi.zmaw.de/index.php?id=432

[4]. Randerson D., Atmospheric science and Power production. Technical Information Center, Office of Scientific and Technical Information. United States Department of Energy. – Vol. 1. – [S.1.], 1994 INTERNATIONAL ATOMIC ENERGY AGENCY, Isotope Techniques in Water Resources Development and Management, C&S Papers Series No. 2/C, IAEA, Vienna (1999) (CD-ROM).

[5]. VDI (2005). Environmental meteorology – Prognostic microscale windfield models – Evaluation for flow around buildings and obstacles. VDI guideline 3783, Part 9. BeuthVerlag, Berlin.

[6]. V.V. Belikov, V.M. Goloviznin, U.V. Katishkov et. all Proceeding of IBRAE RAS/ Ed. By L.A. Bolshov Issue 9 Modeling of radionuclide transport in the environment. Moskow:-Nauka, 2007.-229pp.

[7]. Deardorff D.W., Parameterization of the planetary boundary layer for use in general circulation models, Mon. Wea. Rev., Vol. 100, №2, 1972.

[8]. COST ES1006 – Best practice guidelines, COST action ES1006, april 2015. ISBN: 987-3-9817334-0-2

[9]. Special Issue Joint Urban 2003. Journal of Applied Meteorology and Climatology, Volume 46, Issue 12 (December 2007)

[10]. Jörg Franke, Antti Hellsten, Heinke Schlünzen and Bertrand Carissimo: Best practice guideline for the CFD simulation of flows in the urban environment. COST 732 report, Hamburg, 2007, ISBN: 3-00-018312-4

[11]. A Validation of FEM3MP with Joint Urban 2003 Data Stevens T. Chan and Martin J. Leach Lawrence Livermore National Laboratory, Livermore, California 94551, USA

Специализированная робастная CFD RANS микромасштабная метеорологическая модель для моделирования атмосферных процессов и переноса примеси в условиях городской и промышленной застройки

О.С. Сороковикова <olga_sorokov@mail.ru>
Д.В. Дзама <diman_sw@mail.ru>
Д.Г. Асфандияров <dasfandiyarov@ibrae.ac.ru>
Институт проблем безопасного развития атомной энергетики РАН, 115191, Россия, г. Москва, ул. Большая Тульская, д. 52

Аннотация. В последние годы в мировой практике существует тенденция к использованию специализированных CFD моделей в задачах вычислительной метеорологии. К таким задачам относится, в частности, задача обоснования безопасности промышленных объектов, в том числе радиационно-опасных. Эта тенденция обусловлена тем фактом, что применение универсальных инженерных кодов общего назначения требует изрядных вычислительных мощностей и связано это в первую очередь с необходимостью сгущения расчётной сетки к поверхностям земли и зданий для разрешения вязкого и промежуточного слоёв. С другой стороны, гауссовы модели не могут учесть сложные аэродинамические эффекты, возникающие при обтекании зданий сложной конфигурации, в том числе описать все тонкости обтекания сооружений примесью при атмосферных выбросах газо-аэрозольных веществ. Поэтому авторами была разработана робастная узкоспециализированная CFD-RANS модель и расчётный код для моделирования атмосферной дисперсии примеси в условиях сложной трёхмерной геометрии, не требующие сгущения сеток. Авторы работы провели верификацию этой модели на различных данных, полученных как в ходе натурных крупномасштабных, так и в ходе лабораторных туннельных экспериментов. Для этих целей была использована, в частности, рекомендованная международным экспертным сообществом база данных и соответствующие статистические характеристики соответствия рассчитанных и полученных в ходе экспериментов значения компонент скорости течения и концентрации примеси. Результаты верификации показали, что разработанная модель удовлетворяет приёмочным критериям качества моделирования наравне с зарубежными кодами общего назначения и узкоспециализированными кодами.

Ключевые слова: микромасштабные метеорологические модели; перенос примеси; расчет доз.

DOI: 10.15514/ISPRAS-2018-30(5)-13

Для цитирования: Сороковикова О.С., Дзама Д.В., Асфандияров Д.Г. Специализированная робастная CFD RANS микромасштабная метеорологическая модель для моделирования атмосферных процессов и переноса примеси в условиях городской и промышленной застройки. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 213-234 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-13

Список литературы

[1]. Michael Schatzmann, Helge Olesen, Jörg Franke: COST 732 Model Evaluation Case Studies: Approach and Results. University of Hamburg Meteorological Institute Centre for Marine and Atmospheric Sciences, 2010, http://www.mi.uni-hamburg.de/Official-Documents.5849.0.html

[2]. SoundPlan. MISKAM advanced. http://www.soundplan.eu/english/soundplan-air-pollution/miskam-advanced/

[3]. Stevens T. Chan, Martin J. Leach. A Validation of FEM3MP with Joint Urban 2003 Data. Journal of applied meteorology and climatology. Vol. 46. 2007. pp. 2127-2146 Data base CEDVAL at Hamburg University, http://www.mi.zmaw.de/index.php?id=432

[4]. Randerson D., Atmospheric science and Power production. Technical Information Center, Office of Scientific and Technical Information. United States Department of Energy. – Vol. 1. – [S.1.], 1994 INTERNATIONAL ATOMIC ENERGY AGENCY, Isotope Techniques in Water Resources Development and Management, C&S Papers Series No. 2/C, IAEA, Vienna (1999) (CD-ROM).

[5]. VDI (2005). Environmental meteorology – Prognostic microscale windfield models – Evaluation for flow around buildings and obstacles. VDI guideline 3783, Part 9. BeuthVerlag, Berlin.

[6]. V.V. Belikov, V.M. Goloviznin, U.V. Katishkov et. all Proceeding of IBRAE RAS/ Ed. By L.A. Bolshov Issue 9 Modeling of radionuclide transport in the environment. Moskow:-Nauka, 2007.-229pp.

[7]. Deardorff D.W., Parameterization of the planetary boundary layer for use in general circulation models, Mon. Wea. Rev., Vol. 100, №2, 1972.

[8]. COST ES1006 – Best practice guidelines, COST action ES1006, april 2015. ISBN: 987-3-9817334-0-2

[9]. Special Issue Joint Urban 2003. Journal of Applied Meteorology and Climatology, Volume 46, Issue 12 (December 2007)

[10]. Jörg Franke, Antti Hellsten, Heinke Schlünzen and Bertrand Carissimo: Best practice guideline for the CFD simulation of flows in the urban environment. COST 732 report, Hamburg, 2007, ISBN: 3-00-018312-4

[11]. A Validation of FEM3MP with Joint Urban 2003 Data Stevens T. Chan and Martin J. Leach Lawrence Livermore National Laboratory, Livermore, California 94551, USA

Numerical simulation of motions of a ship with a moonpool in head waves

K.D. Ovchinnikov <ovchinnikov_kd@mail.ru>
Saint Petersburg State Marine Technical University,
3, Lotsmanskaya St., Saint Petersburg, 190121, Russia

Abstract. The paper shows the results of assessing the possibilities of computational fluid dynamics for predicting the motions of a ship with a moonpool and vertical water motions in a moonpool in regular head waves with a zero ship speed. A moonpool is a well which is used in different types of ships such as cable laying, drill and FPSO, survey, research and so on. This well is used for launching and lifting of different devices, divers, rescue bells, cables and risers, which are protected from outboard wind and waves. The results of the numerical simulation in the OpenFOAM software of heave and pitch motions of the DTMB 5415 model in regular head waves with and without ship speed show good agreement with the experimental data. The experiment was organized with a series 60 model, which was equipped with different moonpool shapes modules in regular head waves with a zero ship speed for determining heave and pitch RAOs and vertical water motions in the moonpool. The results do not show any influence of the moonpool for heave and pitch ship motions. These data are necessary for numerical simulation verification. The results of the numerical simulation of experimental research show good agreement, which means the good efficiency of computational fluid dynamics in heave and pitch motions and vertical water motions in moonpool calculation of a ship with a moonpool. Numerical simulation should be advised for calculations during the ship design process.

Keywords: numerical simulation; CFD; experiment; moonpool; heave motions; pitch motions; RAO; DTMB 5415; series 60; OpenFOAM.

DOI: 10.15514/ISPRAS-2016-30(5)-14

For citation: Ovchinnikov K.D. Numerical simulation of motions of ship with moonpool in head waves. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 235-248. DOI: 10.15514/ISPRAS-2016-30(5)-14

1. Introduction

The moonpool is a vertical well that is used on different types of ships, such as cable laying, mining and drilling, rescue, research, supply and support, is called the moonpool. This moonpool is intended for lowering and lifting various equipment, divers or rescue bells, cables or risers protected from external waves.

There are vertical water motions in the moonpool. Usually the amplitudes of water motions are no greater than the amplitudes of incoming waves. In rare cases,

resonant motions can occur, which can be four times larger than the amplitude of the incoming wave, and which can lead to the damage of the ship or the equipment located in the moonpool [1].

An experiment is organized to determine the characteristics of ship motion. The most common method for studying ship motion is forced motions in regular waves, which result RAO and phase lags of the studied types of ship motion [2].

However, the experiment is a complicated and expensive event; therefore, the use of numerical methods to solve the problems of ship hydromechanics is becoming increasingly popular.

Nowadays, methods of computational fluid mechanics (CFD) are widely used for calculating ship resistance and are also actively used for developing approaches to calculations ship motions.

In this paper, the following tasks are solved:

- The verification of CFD methods for calculations of heave and pitch motions RAO and phase lags;
- The experimental research of the motions of the ship with a moonpool without ship speed in head waves;
- The verification CFD methods for calculations of heave and pitch motions and vertical water motions RAO of the ship with moonpool.

The experimental research is needed because there are not any open experimental data of motions of ship with moonpool.

OpenFOAM software is used like CFD methods [3].

2. Preparation of numerical simulation in OpenFOAM

The Navier-Stokes equation for an incompressible fluid is defined as a system of momentum conservation equations and the continuity equation.

At high Reynolds numbers, computations with direct numerical simulations are accompanied by either large time costs or large computational powers; therefore, the time-averaged Navier-Stokes equations, called Reynolds equations, are commonly used to simulate turbulent flows. These equations are derived from the Navier-Stokes equations by splitting the pressure velocity fields into an average value and fluctuation [4].

The two-parameter turbulence model k-ω SST is used to close the system of equations [5].

Discretization of fundamental equations in the OpenFOAM software is performed using the finite volume method; the free surface is simulated using the modified volume of fluid method.

The motion of a solid body with six degrees of freedom can be described by the dynamic Euler equations [6]. The standard interDyMFoam solver was chosen for solving assigned tasks.

The internal utilities of the OpenFOAM package, such as topoSet, refineMesh and snappyHexMesh, are used to create the mesh.

Built-in boundary conditions, such as waveVelocity and waveAlpha for velocity and phase fraction respectively, at the input boundary, and functions like verticalDamping for the fvOptions file in the numerical beach zone that appeared in the OpenFOAM-5.0 version, are used to simulate regular waves.

In order to simulate the ship dynamics at sea, it is necessary to create a high-quality mesh not only in the free surface zone but also on a general scale. The recommended dimensions of the mesh in the numerical simulation of ship dynamics at sea are presented in fig. 1. In fig. 1 L is the length of ship or the wavelength, whichever is greater.

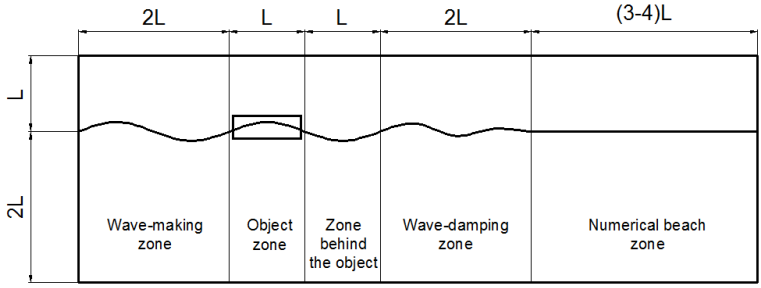

Fig. 1. Recommended dimensions of mesh for numerical simulation of ship dynamics

3. Numerical simulation of heave and pitch motions

According to the recommendations of ITTC, the study of ship motions at regular waves should be carried out under the following conditions [7]:

- the ratio of wave height to its length or wave height to model length should be constant (the recommended ratio of wave height to its length is about 1/50);
- wavelengths should vary in the range from 0.5 to 2 lengths of the model under study;
- the optimal number of motions in tests for analyzing the motion characteristics is not less than 10.

The heave and pitch motions were simulated for the DTMB 5415 model (a ship of the Arleigh Burke-class, US Navy). Experimental data used to verify the results of numerical simulations are presented in [8].

Characteristics of the DTMB 5415 hull: length between perpendiculars L_{pp} = 3.048 m, width B = 0.409 m, draft T = 0.132 m, displacement D = 83.5 kg, vertical center of gravity z_g = 0.163 m, moments of inertia J_{44} = 1.92 kg·m^2 and J_{55} = 48.5 kg·m^2.

The simulation was performed at two ship speeds corresponding to Froude numbers of Fn = 0 and Fn = 0.28. Head waves was simulated. The model had only two degrees of freedom – heave and pitch motions.

It was created the mesh with an axis of symmetry possessing the following characteristics: the number of cells was ~ 1.5 million, the maximum proportionality coefficient was ~ 57, the maximum twist factor was ~ 2.4, the average non-orthogonality was ~ 4.3, the average value of the dimensionless characteristic was y+ ≈ 1.

All calculations were made on the computer cluster of the Department of Hydro and Aeromechanics and Marine Acoustics and the Department of Applied Mathematics and Mathematical Modeling of St. Petersburg State Marine Technical University.

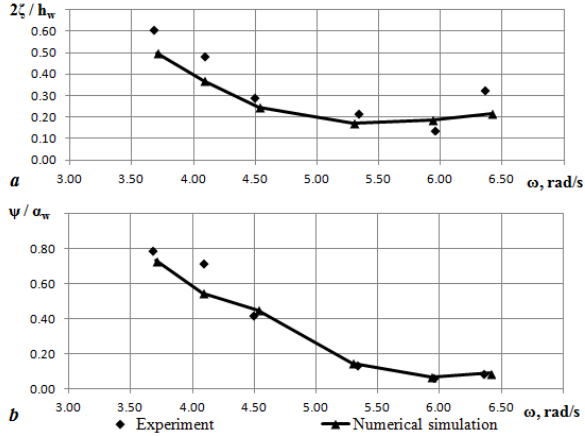

Fig. 2. RAO of heave (a) and pitch (b) motions without ship speed.

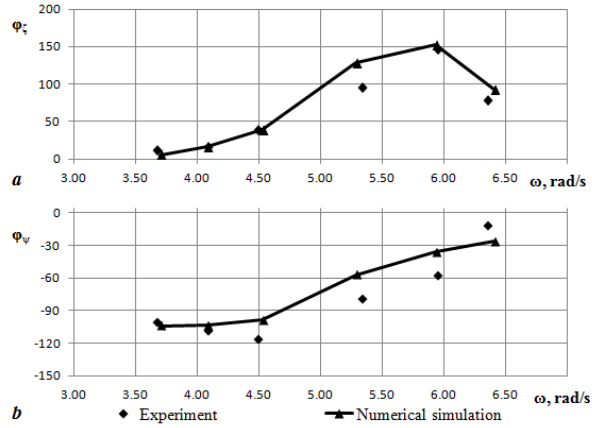

Fig. 3. Phase lags of heave (a) and pitch (b) motions without ship speed

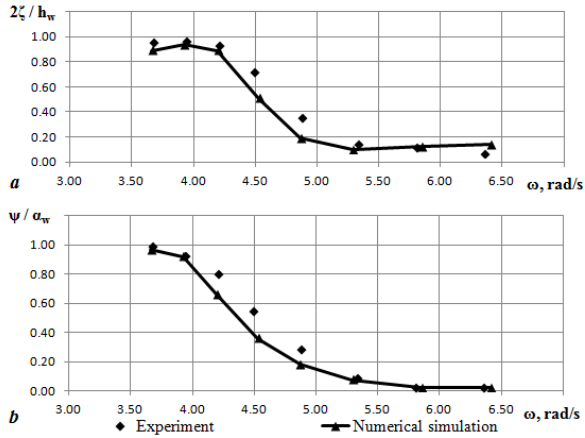

Fig. 4. RAO of heave (a) and pitch (b) motions with Fr = 0.2

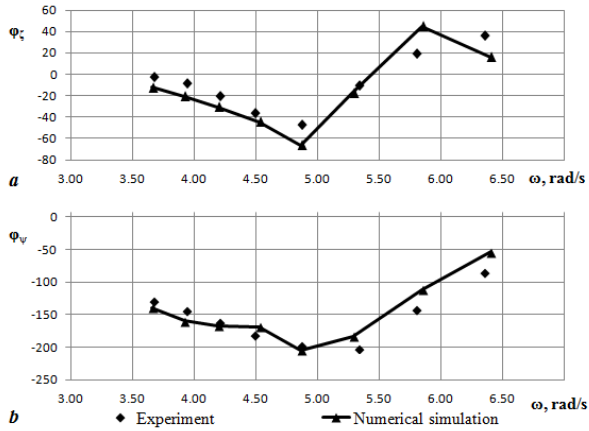

Fig. 5. Phase lags of heave (a) and pitch (b) motions with Fr = 0.28

According to the simulation results, the RAO and phase lags of heave and pitch motion are obtained and presented in figs. 2-5. In figs. 2-5, the following notation is used: $2\zeta/h_w$ – dimensionless amplitude of the heave motions of the ship (ratio of absolute heave motions ζ to the amplitude of incoming waves $h_w/2$, where h_w – height of the incoming waves), φ_ζ – the phase lags of heave motions of the ship, ψ/α_w – dimensionless amplitude of the pitch motions of the ship (the ratio of

absolute pitch motions ψ to the angle of the incoming wave α_w), φ_ψ – the phase lags of pitch motions of the ship, ω – the frequency, rad/s.

Analyzing figs. 2-5, it can be concluded that numerical simulation allows estimating well enough the RAO and phase lags of heave and pitch motion both without and with ship speed.

At the same time, there is no need for additional calculations to assess the effect of grid convergence since the obtained results satisfy engineering accuracy, and the speed of calculation (about 24 hours for one RAO and phase lags) allows using this approach in the different stages of ship design process.

4. Experimental research of motions of ship with moonpool

The place of the research is the towing tank of the Ship Theory Department of the St. Petersburg State Maritime Technical University.

A plunger wave producer is used to create the incoming waves. The installed wave producer can create only two-dimensional regular waves.

In order to perform the experiment, the series 60 model was created with the following characteristics: length between perpendiculars L_{pp} = 2.09 m, width B = 0.289 m, draft T = 0.125 m, displacement D = 45 kg. In order to study the effect of the presence of moonpool, a model was created with a modular insert in the longitudinal center of buoyancy. Three modules have been developed:

- module No. 1 – no moonpool;
- module No. 2 – a circular moonpool with an internal diameter of d_m = 0.044 m and a relative diameter of d_m/B = 15%;
- module No. 3 – a moonpool of the circular cross-section with an inner diameter d_m = 0.074 m and a relative diameter d_m/B = 25%.

Three-dimensional models of the ship with different modules are presented in Fig. 6.

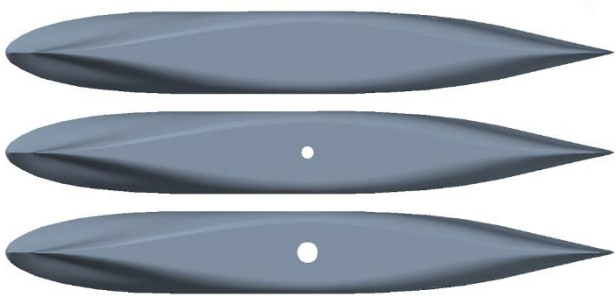

Fig. 6. Bottom view for 3D models with different moonpool modules: upper – module No. 1, middle – module No. 2, lower – module No. 3

The following model characteristics were obtained based on the results of static and dynamic calibrations: the longitudinal center of gravity from the midsection x_g = -0.03 m, the transverse center of gravity y_g = 0.00 m, the vertical center of gravity z_g = 0.10 m, the moments of inertia J_{44} = 0.8 kg·m^2 and J_{55} = 6.5 kg·m^2.
The model was tested in regular head waves with length ranging from 1.5 to 4.0 m. The image of the motions of the model on the 4.0 m wave is shown in fig. 7.

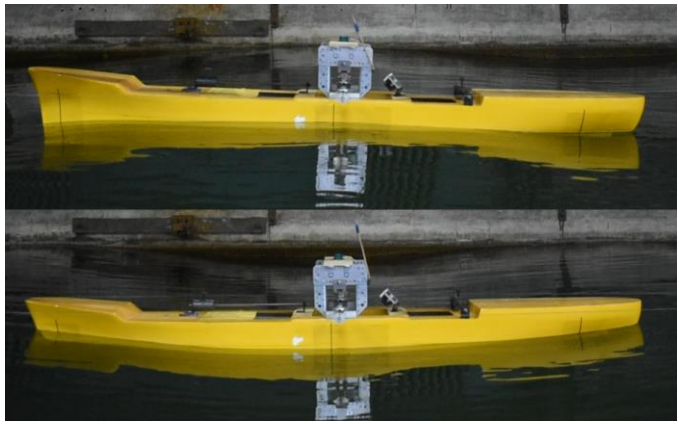

Fig. 7. Ship motions in head waves with a length of 4.0 m

As the experimental results, the heave and pitch motions data were obtained, as well as the vertical water motions in moonpool. After processing the data, RAO of heave and pitch motions of the model with different modules, as well as the vertical water motions in moonpool, which are shown in Figs. 8-10 respectively. In Figs. 8-10, approximating lines for each RAO are additionally presented.

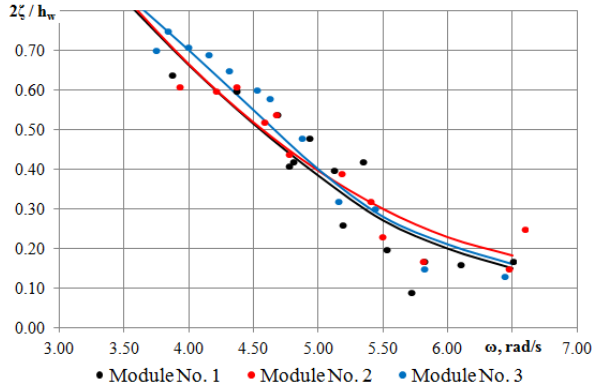

Fig. 8. RAO of heave motions for the model with different moonpool modules.

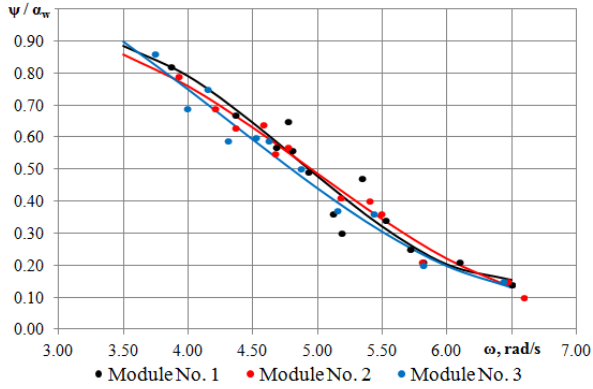

Fig. 9. RAO of pitch motions for the model with different moonpool modules.

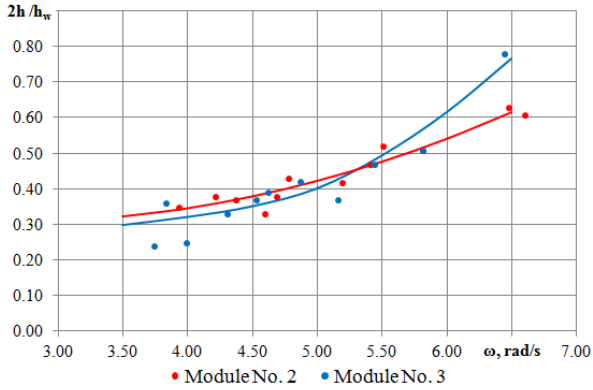

Fig. 10. RAO of water vertical motions in the moonpool for the model with different moonpool modules.

Analyzing the RAO presented in figs. 9 and 10, it can be concluded that the presence of a circular moonpool with a diameter up to 25% of the ship width does not affect the heave and pitch motions.

5. Numerical simulation of motions of ship with moonpool

In order to perform numerical simulations, three-dimensional models of the series 60 ship shown in fig. 6 were developed.

In order to study the heave and pitch motions, meshes were created with an axis of symmetry with the number of cells ranging from 1 to 1.5 million, depending on the wavelength. The characteristics of meshes are similar to those obtained in the numerical simulation of the DTMB 5415 hull.

According to the results of the numerical simulation, RAO of the heave and pitch motions of the ship, as well as the vertical water motions in moonpool were obtained and are presented in figs. 11-13 for models with modules No. 1-3, respectively.

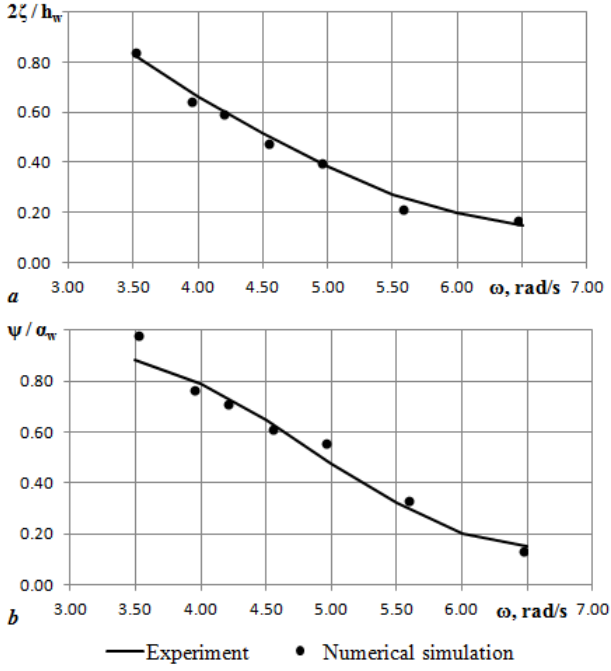

Fig. 11. RAO of heave (a) and pitch (b) motions of the model with module No. 1.

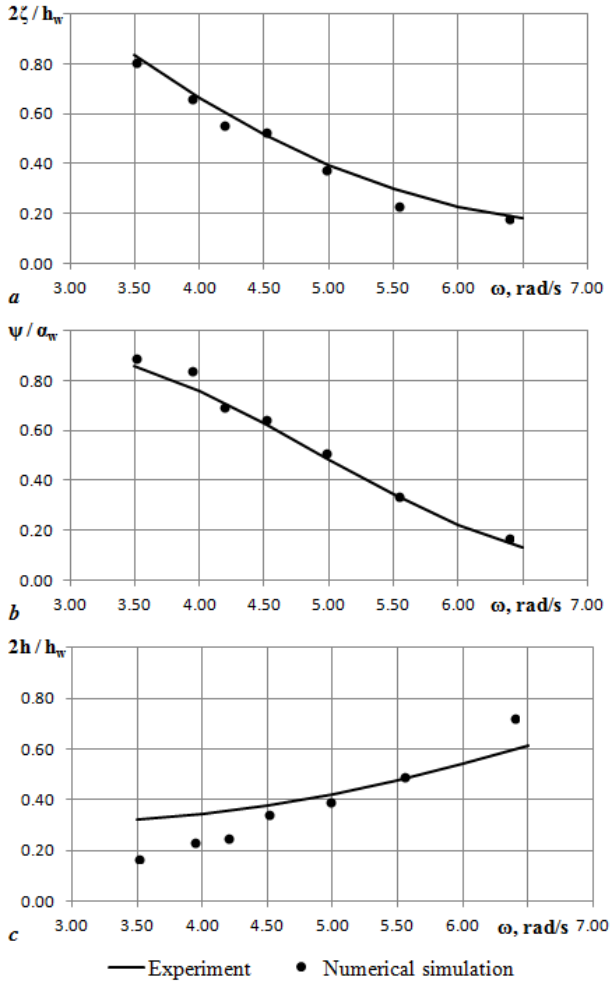

Fig. 12. RAO of heave (a) and pitch (b) motions and vertical water motions in the moonpool (c) of the model with module No. 2.

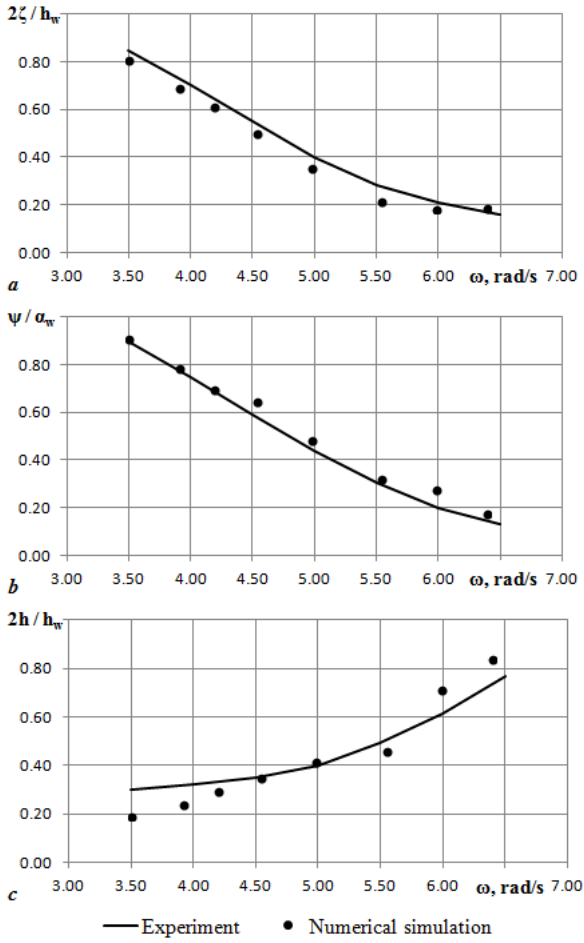

Fig. 13. RAO of heave (a) and pitch (b) motions and vertical water motions in the moonpool (c) of the model with module No. 3.

Upon analyzing the data presented in figs. 11-13, it can be concluded that numerical simulation allows determining the characteristics of the heave and pitch motions of ships with moonpools of different diameters in head sea with good accuracy.

In this case, attention should be paid to figs. 12c and 13c, which present RAO of the vertical water motions in moonpool. It can be noted here that in the considered frequency range, numerical simulation allows determining the motions characteristics of the water in moonpool with sufficiently good accuracy. However, in fig. 12b, there are differences in the trends of numerical simulation and the experimental results, especially during subsequent extrapolation to the high-

frequency zone. This remark is unfair for the results of the model with module No. 3 simulation.

Numerical simulation of the motion of ship with moonpool can be recommended in the different stages of ship design process for determining the characteristics of ship motions and the vertical water motions in moonpool.

6. Conclusion

The paper shows the results of assessing the possibilities of computational fluid dynamics for predicting the motions of a ship with a moonpool and vertical water motions in a moonpool in regular head waves with a zero ship speed.

The results of the numerical simulation in the OpenFOAM software of heave and pitch motions of the DTMB 5415 model in regular head waves with and without ship speed show good agreement with the experimental data.

The experiment was organized with a series 60 model, which was equipped with different moonpool shapes modules in regular head waves with a zero ship speed for determining heave and pitch RAOs and vertical water motions in the moonpool. The results do not show any influence of the moonpool for heave and pitch ship motions. These data are necessary for numerical simulation verification.

The results of the numerical simulation of experimental research show good agreement, which means the good efficiency of computational fluid dynamics in heave and pitch motions and vertical water motions in moonpool calculation of a ship with a moonpool. Numerical simulation should be advised for calculations during the ship design process.

Numerical simulation of the motion of ship with moonpool can be recommended in the different stages of ship design process for determining the characteristics of ship motions and the vertical water motions in moonpool.

References

[1]. Guilhem Gaillarde and Anke Cotteleer, Water motion in moonpools empirical and theoretical approach. Maritime Research Institute Netherlands MARIN, 2001.

[2]. Borisov R.V., Semenova V.Y. Ship motions. SPb.: Publ. SMTU, 2015. 93 p. (in Russian).

[3]. https://openfoam.org/ (01.09.2018).

[4]. Henry Peter Piehl. Ship Roll Damping Analysis. Von der Fakultät für Ingenieurswissenschaften, Abteilung Maschinenbau und Verfahrenstechnik, der Universität Duisburg-Essen zur Erlangung des akademischen Grades eines. Doktors der Ingenieurswissenschaften Dr.-Ing. April 2016.

[5]. Menter, F. R. Two-equation eddy-viscosity turbulence models for engineering applications. AIAA Journal, vol. 32, No. 8, 1994, pp. 1598-1605.

[6]. Ovchinnikov K.D., Tryaskin N.V., Tkachenko I.V. Numerical simulation of semi-submersible rig motions in regular waves. Marine intellectual technologies. № 2 (28) V.1 2015 (in Russian).

[7]. ITTC – Recommended Procedures and Guidelines. Seakeeping Experiments. 7.5-02 07-02.1. 2014.
[8]. Irvine, M., Longo, J. and Stern, F. Pitch and Heave Tests and Uncertainty Assessment for a Surface Combatant in Regular Head Waves, Journal Ship Research, vol. 52, No. 2, June 2008, pp. 146-163.

Численное моделирование качки судна с шахтным устройством на встречном волнении

К.Д. Овчинников <ovchinnikov_kd@mail.ru>
Санкт-Петербургский государственный морской технический университет, 190121, Санкт-Петербург, ул. Лоцманская, 3

Аннотация. В работе приводятся результаты оценки возможности применения современных средств вычислительной гидромеханики для определения характеристик качки судна с шахтным устройством и колебаний жидкости в шахте на встречном волнении при отсутствии скорости хода. Шахтой называется «колодец», используемый на различных типах судов, таких как, кабелеукладочных, добычных и буровых, спасательных, исследовательских, снабжения и обеспечения. Эта шахта предназначена для спуска и подъема различного оборудования, водолазов или спасательных колоколов, кабелей или райзеров, защищенных от воздействия внешнего волнения. Результаты численного моделирования в программном комплексе OpenFOAM продольной качки модели DTMB 5415 на встречном регулярном волнении при наличии и отсутствия скорости хода показали, что численное моделирование позволяет с высокой эффективностью определять амплитудно-частотные и фазово-частотные характеристики вертикальной и килевой качки судна. Проведенное экспериментальное исследование модели серии 60 без шахты и с двумя шахтами круглого сечения различного диаметра на встречном волнении при отсутствии скорости хода позволило получить амплитудно-частотные характеристики вертикальной и килевой качки, а также вертикальных колебаний жидкости в шахте, которые показали, что влияние наличия шахты на динамику судна на встречном волнении пренебрежимо мало. Эти данные также необходимы для выполнения верификации численного моделирования колебаний судна с шахтой на встречном волнении. Численное моделирование экспериментального исследования показало, что современные средства вычислительной гидромеханики позволяют с хорошей эффективностью решать задачи по определению характеристик продольной качки судна, снабженного шахтным устройством, и вертикальных колебаний жидкости в шахте на встречном волнении при отсутствии скорости хода. Численное моделирование качки судна с шахтным устройством может быть рекомендовано на начальных стадиях проектирования для определения параметров качки и вертикальных колебаний жидкости в шахте.

Ключевые слова: численное моделирование; эксперимент; шахта; вертикальная качка; килевая качка; амплитудно-частотная характеристика; DTMB 5415; серия 60; OpenFOAM.

DOI: 10.15514/ISPRAS-2018-30(5)-14

Для цитирования: Овчинников К.Д. Численное моделирование качки судна с шахтным устройством на встречном волнении. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 235-248 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(5)-14

Список литературы

[1]. Guilhem Gaillarde and Anke Cotteleer, Water motion in moonpools empirical and theoretical approach, Maritime Research Institute Netherlands MARIN, 2001.

[2]. Борисов Р.В., Семенова В.Ю. Качка корабля: учеб.пособие. Под ред. д-ра техн. наук, проф. Р.В. Борисова. СПб.: Изд-во СПбГМТУ, 2015, 93 с.

[3]. https://openfoam.org/ (01.09.2018).

[4]. Henry Peter Piehl. Ship Roll Damping Analysis. Von der Fakultät für Ingenieurswissenschaften, Abteilung Maschinenbau und Verfahrenstechnik, der Universität Duisburg-Essen zur Erlangung des akademischen Grades eines. Doktors der Ingenieurswissenschaften Dr.-Ing. April 2016.

[5]. Menter, F. R. Two-equation eddy-viscosity turbulence models for engineering applications. AIAA Journal, vol. 32, No. 8, 1994, pp. 1598-1605.

[6]. Овчинников К.Д., Тряскин Н.В., Ткаченко И.В. Численное моделирование качки полупогружной платформы на регулярном волнении. Морские интеллектуальные технологии, № 2 (28) т.1 2015.

[7]. ITTC – Recommended Procedures and Guidelines. Seakeeping Experiments. 7.5-02 07-02.1. 2014.

[8]. Irvine, M., Longo, J. and Stern, F. Pitch and Heave Tests and Uncertainty Assessment for a Surface Combatant in Regular Head Waves, Journal Ship Research, vol. 52, No. 2, June 2008, pp. 146-163.

Ontological CFD-repository

V.A. Zenkin <vl.zenkin@gmail.com>
Bauman Moscow State Technical University
5/1, 2-nd Baumanskaya st, Moscow, 105005, Russia

Abstract. Based on the RDF-storage, a software tool was developed for creating a knowledge base containing information about the CFD-calculations performed. The software is a set of scripts written in bash and python, which are published under the GNU GPL3 license. The tool is designed to support the user when making research studies that do not have a strict, pre-defined design of experiment or problem solving algorithm. To formalize the description of the calculation stored in the knowledge base, an ontology is created that serves as the information model for the calculation. As an auxiliary mechanism for carrying out an automated comparison of calculations with each other (a mechanism of "comparators" and "features") was developed and also described in the article. In addition to the systematized data storage, the complex provides the possibility of their automated and semi-automated analysis, including the presentation of a set of calculations in the form of an undirected graph, the construction of flat and spatial dependencies, the search for similar calculations, etc. The article gives examples of data processing results for the project on design of channel in the cylinder head of a piston engine.

Keywords: CAE methodology; ontology; knowledge management; computational gas dynamics; semantic technologies

DOI: 10.15514/ISPRAS-2016-30(5)-15

For citation: Zenkin V.A. Ontological CFD-repository. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 249-264. DOI: 10.15514/ISPRAS-2016-30(5)-15

1. Introduction

At the initial stage of numerical research during Computer-Aided Engineering (CAE), the typical situation is intuitive search in a multidimensional factor parameter space. The purpose of this search is to obtain primary knowledge about a system. At this stage, it is almost impossible to produce a rigorous plan of experiment, thus, it is often unsystematic and extremely difficult to process and analyze its data. Usually, all results are presented in a tabular form (an example of the table of CFD calculations from the experience of the author of this article is shown in fig. 1), which is often very poorly systematized (because the system is just the result of this work). The analysis of such tables becomes very difficult when the number of calculations exceeds several dozen. It results in an increase in the complexity of this research stage, frequent excess and repeated calculations, which can be avoided with proper systematization of obtained results.

	A	B	C
1	---------- Profiling the exhaust channel ----------	G, kg/s	
2	base — base channel for analysis, in 2 sections, with a lift of 7 mm. For him, the analysis of grid converge	0,155	0,0%
3	base_without_bob — it's without a lug	0,155	0,0%
4	base_without_styk - it's without a groove at the junction of the channel and pipe	-	
5	base_without_ustup - it's without a step between the saddle and the channel	0,156	0,6%
6	----------		
7	opt7 - the geometry formed for optimization on lift of 7 mm. Compared with the base cleaned ledge, lug, jo	0,155	0,0%
8	opt7_b - it's with an increased output section to R19	0,156	0,6%
9	opt7_b_wps - it's with an increased output section to R19 but without an intermediate section	0,157	1,3%
10	opt7_g - it's with a narrowed intermediate section	0,147	-5,2%
11	opt7_gb - it's with a narrowed intermediate and increased output	0,148	-4,5%
12	opt7_rb - it's also with extended intermediate and extended output	0,157	1,3%
13	opt7_eb - it's with an ellipse of 33 to 38 in the intermediate section (a little more than the standard) and an	0,156	0,6%
14	opt7_e36 - ellipse as above. Output section R18 (between standard and b)	0,156	0,6%
15	opt7_b_wps_E - it's with an increased output section to R19 in the shape of an ellipse and without an inter	0,1575	1,6%
16	---------- Checking other pressure drop		
17	------- At a drop of 1,1 to 1		
18	opt7	0,0287	0,0%
19	opt7_rb	0,0297	3,5%
20	------- At a drop of 5 to 1		
21	opt4	0,132	0,0%
22	opt4_rb	0,1325	0,4%

Fig. 1. Example of the table with the results of the initial search study of the engine exhaust channels (the initial stage of channel profiling)

The problem of scientific and methodological support for an engineer and researcher in the field of knowledge management (accumulation, structuring, reuse, automatic and semi-automatic analysis) has recently been the subject of a large number of works. This growth rate in publications and papers is associated with the development of semantic technologies (methods and tools that provide and use information coding, in which the value is stored separately from data) and with the emergence of protocol standards and file standards, and the development of supporting software.

If one considers publications only about Computer-Aided Engineering (CAE), the most important works are on user support when carrying out numerical modeling (due to the relatively high complexity of this procedure); in addition, these technologies can be used for configuration problem solving [1,2].

In the work [3], the authors solve the problem of experiment planning. For this purpose, they created an ontology – an explicit formal description of terms in a domain and the relationships among them – and a data warehouse of the performed optimization procedures together with the results of their work. An engineer uses this set as a knowledge base when choosing parameters to plan a numerical experiment. In the work [4], an ontology and semantic knowledge base are used for the logical processing of constraints imposed on structural components of construction, which should reduce the number of errors in solving a design problem. In the work [5], the use of standardization technologies for strength calculations promises a 75% reduction in the complexity of similar tasks. The author of [6, 7] works on the application of these technologies in the aviation and space industry. In

addition, there are many examples of the use of ontologies and other semantic technologies in other areas included in the product lifecycle.

Currently, there are many software tools designed to systematize and accumulate knowledge. They can be both general (PDM systems) and narrowly focused on the creation of CAE knowledge bases [8, 9], but these products have a predominantly corporate purpose and are intended, first of all, to reuse the results of calculations (including by other calculation specialists) and to organize work in a large team. This is expressed in the fact that final calculations are saved in such systems, and intermediate (erroneous or incomplete) calculations are not recorded; however, sometimes they contain no less important information for a specialist. The author of this article was unable to find a single software product designed to support the systematization of calculations for the individual work of a calculation specialist that solves a task without any known algorithms in advance. (An example of a similar product could be the popular git version control system, which provides access to a program code at all stages of its development at once).

Thus, the purpose of this article is to create a tool that solves the following problems:

- collecting and storing information about calculations;
- presenting this information in a structured (corresponding to a relational model) or semi-structured (containing labels for separating semantic elements and for ensuring a hierarchical structure of records) way, i.e. not as raw data, but in the form of ordered knowledge;
- processing and presenting the collected knowledge to a user in a way that simplifies analysis, or that helps to perform this analysis automatically.

The field of application of the developed tool is performing calculations in Computational Fluid Dynamics (CFD); however, the proposed method can be easily adapted to other areas of CAE.

2. Storage of Repository of Calculations in a Knowledge Base

This article proposes to use a semantic knowledge base in the RDF format [10], which can be accessed via a SPARQL. This base is for storing information about the performed calculations. The RDF format stores data in a triple format, i.e. statements of the following type “subject-predicate-object” (for example, “calculation1 obtained_with_program simpleFoam”. In addition, each entity appearing in the knowledge base may be mentioned an unlimited number of times and in any of these positions. As a result, the entire knowledge base is a directed graph of arbitrary structure. Existing SPARQL implementations make a graph search more efficient using a query language similar to SQL for usual databases.

To consider a semantic database as a knowledge base, in addition to direct information about specific calculations (ABox), it must also contain an information model defining the structure of this information and its semantics (TBox).

An information model is a formal model of a limited set of facts, concepts or instructions designed to meet a specific requirement [11]. In other words, such

model sets a formal data structure by defining the relationship among data and, thereby, by transforming this data into knowledge. As a rule, the construction of this model is based on mathematical logic. To process and present data (especially numerous and complexly structured), the formation of an information model is as necessary as the formation of a mathematical model for performing calculations. Moreover, as a mathematical model, an information model is verified by its practical application and compliance with the requirements imposed on it. An unsuccessful information model is hard for the perception of information by a man and makes it difficult or even impossible for machine processing.

Currently, the most common way to represent information models is ontologies. They help to select classes of objects and define their interdependence, i.e. at the same time they determine the syntax of a knowledge base by fixing the key names of concepts and relationships and making their logical connections among themselves, thus, they make the work of an inference machine possible.

In this article, the information model of CFD calculations in the form of ontology was proposed. The description of this ontology is presented in the next section.

3. Ontology of CFD Calculations

The structure diagram of the key elements of the developed ontology is shown in fig. 2. The ontology file is a part of the repository and is loaded into its database when a server is started, but it can also be used by itself. Currently, all names of classes and relationships are recorded in Russian for the convenience of a Russian-speaking user, but the author is working on an English version with the translation of all relationships between languages.

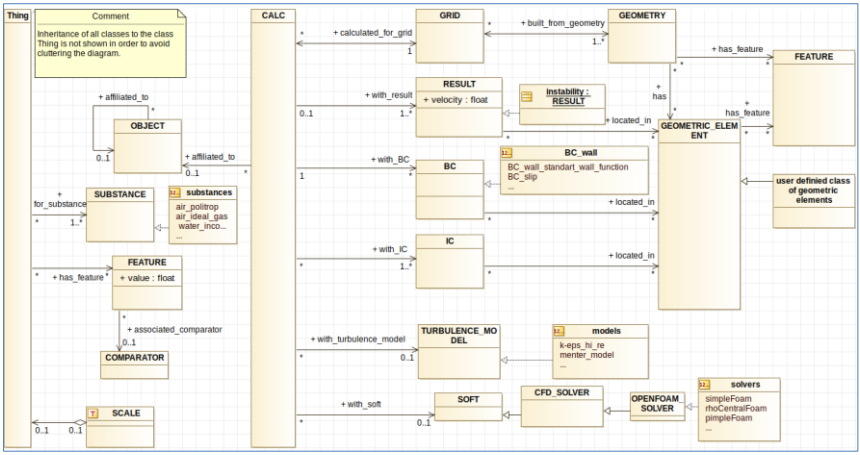

Fig. 2. Ontology of CFD calculations

The central part of this ontology is the mutual relationship between the "Calculation", "Grid" and "Geometry" classes, which correspond to the standard methodology of CFD calculations. Geometry can include the hierarchy of

components – “Geometric Elements”. Each object can contain a number of parameters as defined by the ontology (turbulence model, solver, etc.) or by a user. In this case, no adjustments of the ontology for the repository are not required. For example, to set the “diameter” parameter, use the following line:

```
:geometry_a :diameter "0.1" .
```

where `geometry_a` – a corresponding instance of the `geometry` class. Despite the undoubted importance of the flexibility of data entry, which provides the ability to create the arbitrary hierarchy of components of calculation elements, in addition to this flexibility, the efficiency of recording is of great practical importance, because it directly affects the time, which a user spends when working with the repository. From this point of view, the proposed ontology may seem to be over complicated and the alternative option should be considered, where instead of the “hierarchical” data specification, meaningful prefixes are used. Table 1 shows an example of one object set in these two ways.

Table 1. Comparison of hierarchical and prefix setting of object parameters

Representation	Example of setting a group of objects in a turtle format
Hierarchical representation	:channel_a :contains: input_section_of_channel_a. :input_section_of_channel_a a :Input_section ; :shape :circle ; :area “0.1” .
Hierarchical representation with an anonymous node	:channel_a: contains [a :Input_section; :shape :circle ; :area “0.1”] .
Prefix representation	channel_a :input_section_shape :circle ; :input_section_area “0.1” .

Where :Input_section – user’s geometric class.

The first two representations are formally more stringent and better fit into the concept of a semantic database; in addition, they help to perform accurate addressing to the geometry of other important calculation parameters, for example, boundary conditions. Therefore, the developed ontology is based on the first representation. However, one should point out that they require a longer recording, more difficult in the formulation and perception, thus, in some tasks they may be redundant for a user. Therefore, the structure of concepts was constructed in the way that a user can use the last prefix representation if there is a need. The use of both representations within the same repository is undesirable, but possible.

The other ontology elements generally duplicate the standard structure of a CFD calculation and do not require special comments with the exception of a comparator mechanism, which will be discussed in the following sections. The idea of a function object was presented in order to enable the classification of calculations according to their application area from the point of view of a specific research

object or an ongoing project if a user is going to store several projects in one repository.

4. Ontology Repository Architecture

To demonstrate the capabilities of semantic storage of data on numerical simulation results, the author has created a software package consisting of a set of scripts providing input and processing of data stored in the repository.

In the use-case diagram in fig. 3, typical tasks of a calculation engineer are shown. Elements of the work of an engineer are green (I), which, when using the proposed methodology, are becoming simpler due to a structured knowledge base. Red color (II) – additional tasks that occur when working with the system. Yellow (III) – tasks that are modified when working with the repository. The system, despite the fact that it requires some effort for its customization and development, provides substantial support in the most complex issues of analysis of the calculation bank.

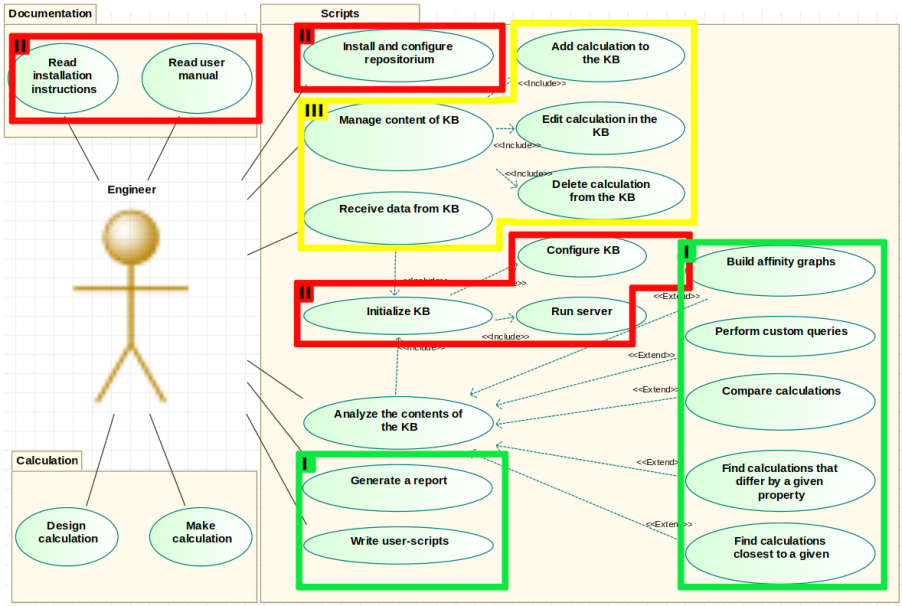

Fig. 3. Use-case diagram for CFD repository

The maintenance of the database of the performed calculations accompanies the work in any case, but the proposed repository provides an alternative tool for this. Unlike many trivial ones (text and table files), it provides the capabilities of automatic and semi-automatic data analysis. In this case, the procedure for tracking data is complicated. For this reason, the use of the repository is expedient in case of a relatively small number of calculations (dozens) combines with their hard-formalized relationships with each other. For example, this includes identification problems, various search studies without a known algorithm, etc. When solving

typical problems or performing a large number of automated calculations that implement factor experiments, the expediency of using the repository is questionable.

Currently, the repository has several tools that solve typical problems:

- comparison of two calculations with each other and selection of similar and different properties;
- construction of two-dimensional and three-dimensional graphs;
- search for two most similar calculations that are different in a given parameter;
- search for calculations that are as close as possible to each other in their parameters;
- construction of an undirected graph containing a complete bank of calculations (or its fragment), whose branches connect calculations that are similar to each other.

The semantic representation of information helps to expand this list indefinitely based on the tasks of a particular user by adding scripts that access the knowledge base. The use of the SPARQL query language, which allows receiving answers to very complex questions from the knowledge base, opens up the widest capabilities for a user. The amount of time required to understand this language (at a basic level) does not exceed several days.

In future, the availability of the knowledge base can be used to solve more complex problems, for example, providing functions to expert systems and decision support systems using an inference machine to the existing ontology and information. However, the mechanisms necessary for this still require their development.

5. Features and Comparators

As can be seen in fig. 3 and the described list of implemented scripts, the key issue in repository work is the problem of comparing two calculations with each other.

To compare, the ontology applies the concept of features and comparators. A feature is a special entity automatically generated in a knowledge base. Each object has a set of such features, and it is assumed that the more differences these lists contain, the greater the number of parameters that objects differ in from each other, and, consequently, the more they are separated from each other. Features are inherited by Geometry -> Grid -> Calculation; in other words, any feature of geometry is a feature of all calculations performed according to this geometry.

A user does not set these features directly, because they are not included in the repository, but they are the result of its processing. Instead, a user assigns corresponding comparators to the parameters of his interest. This approach provides the ability to enter into the knowledge base all available data, because it will be possible not to specify comparators for parameters that are not relevant for a given task, and they will be excluded from automatic analysis procedures.

A comparator is a special label object, which means that an inference machine must automatically generate a feature for all related triplets by a given rule. Currently, five comparators with different feature generation algorithms are implemented. In this case, complex comparators take into account the scale value that a user can set for objects when generating features. The scale is inherited by Functional object -> Calculation -> Grid -> Geometry, which simplifies to set it for a group of similar calculations.

The simple comparator written in the form of a predicate logic formula is given below. Fig. 4 shows an example explaining the second part of the formula that is designed to work with an unnamed first-level node.

$$\forall s,p,o\left(\begin{array}{c} associated\_comparator(p, simple\_comp) \wedge p(s,o) \wedge uri(s) \\ \Rightarrow has\_feature(s, feature\_generated\_comp(p, o)) \end{array}\right)$$

$$\vee$$

$$\forall par,s,\; p,o,type\left(\begin{array}{c} associated\_comparator(p, simple\_comp) \\ \wedge p(s,o) \wedge type(s) \wedge has(par,s) \wedge blankNode(s) \\ \Rightarrow has\_feature(par, feature\_generated\_comp(p, o, type)) \end{array}\right)$$

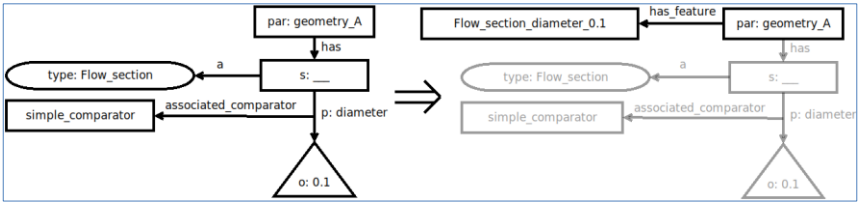

Fig. 4. A diagram illustrating the principle of operation of a comparator with a blank node

The use of the scale mechanism and comparator mechanism potentially make it possible to carry out the comparative analysis of calculations relating to different objects by reducing comparison criteria to dimensionless complexes.

6. Technical Implementation of the Repository

In the current version, the repository is a set of scripts written in bash and python, which interact (using a number of system and third-party programs) with a user and data contained in the repository. Data is stored as text files in the turtle format. To process them, these files are uploaded to the local Apache Fuseki server [12], which provides access to them via a SPARQL. The repository is managed via the command line. To visualize data, a browser (to display html files), the graphviz graph builder with its xdot, gnuplot shell and direct data output to the console are used.

The ontology described in the previous sections is also loaded into Fuseki; however, universal inference machines are not used, because they are not currently required to provide the repository functionality. Instead, a simplified inference machine based

on user rules is used. The comparator mechanism is implemented as separate python scripts.

Simplified work of a user with the repository can be represented as follows.

First, it is necessary to initiate the repository, i.e. start the Fuseki server and enter data into it. The line:

```
> rep start
```

The next step is work with the repository.

To add new data:

```
> rep add sample.ttl
> rep add
```

To display repository content:

```
> rep diplay all calculations
> rep display calculation_original
```

To compare and analyze various data:

```
> rep diagram
> rep compare calculation_original calculation_a
```

If new data has been entered before the analysis, it is necessary to re-initiate the system of comparators:

```
> rep comparators
```

After work is finished, a server can be stopped by the command

```
> rep stop
```

To ensure reliability in terms of the safety of user data at the current stage of development of a software product, no data changes that occur in Fuseki affect the actual file storage of data. Thus, possible script errors or erroneous user requests to the server do not spoil user data.

7. Usage Example

In the last section of this article, the author provides a practical example of using the repository to solve the applied problem of profiling channels in the cylinder head of a piston engine.

On the one hand, this task is characterized by a large number of independent parameters that describe the complex spatial shape of a cast channel, on the other hand, by very small boundaries of parameter changes according to the design considerations, customer requirements, and common sense. (Sometimes there is no reason to change the existing design – when the design documentation has been already prepared, and the production process has already been adjusted – if the potential positive effect is not too great). In such conditions, the formal plans of experiment are of little use, and the profiling process is intuitive and unsystematic at least at the initial stage.

In the described case, the steady state flow of exhaust gases through the exhaust port of a head with two valves of a piston engine with a fixed valve position is shown in fig. 5. The supersonic pressure differential was used as boundary conditions,

because most exhaust gases leave a cylinder in such flow conditions for this engine. The purpose of the work was to increase the flow through the channel. The rigid dimensional requirements and inadmissibility of redesigning the gas air duct determined the boundary narrowness of the permissible change in geometry, which can be provided in various ways. The shape and area of the outlet section from the head, the shape and area of the intermediate section in the middle of the channel, as well as a number of structural elements, were used as main parameters to variate. Also, several additional valve lifts were considered and grid convergence was evaluated. To simulate the flow, the NSF-3 software package was used, which is based on the modification of the method of large particles.

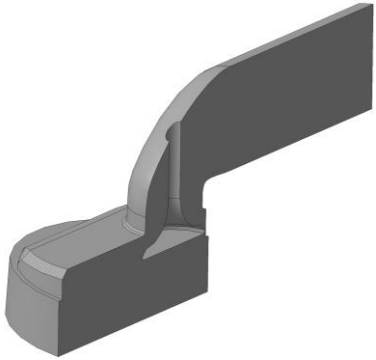

Fig. 5. 3D-model of the computational domain (cutting)

Fig. 1 shows the table of the calculations that records this initial stage of profiling. It is not difficult to notice that, despite its small size, the analysis of this table is very difficult. To test the work of the repository, the data presented in it were converted to the turtle format and uploaded to the repository. An example of the calculation description is as follows:

```
:c1_opt7_b
:mass_flow "0.156";
:obtained_with_program :NSF_2017;
:ted_to :single_exhaust_port;
:for environment: air_politropic;
:with_boundary_conditions
               [:located_in :Output_section;
                :pressure 100000];
:with_boundary_conditions
                     [:located_in :Cylinder_cut;
                    :total_pressure 300000;
                    :total_temperature 1300];
:calculated_for_grid: gr_opt7_b
.
:gr_opt7_b
:built_from_geometry:g_opt7_b
.
```

```
:g_opt7_b
:valve_lift "0.007";
:contains[a :Output_section;
            :nominal_diameter 38;
            :shape :circle];
:conrains [a :Intermediate_section;
             :nominal_diameter 35;
             :shape :circle];
.
```

Figs. 6, 7, and 8 show the results of processing these data using scripts with minimal prepress of screenshots.

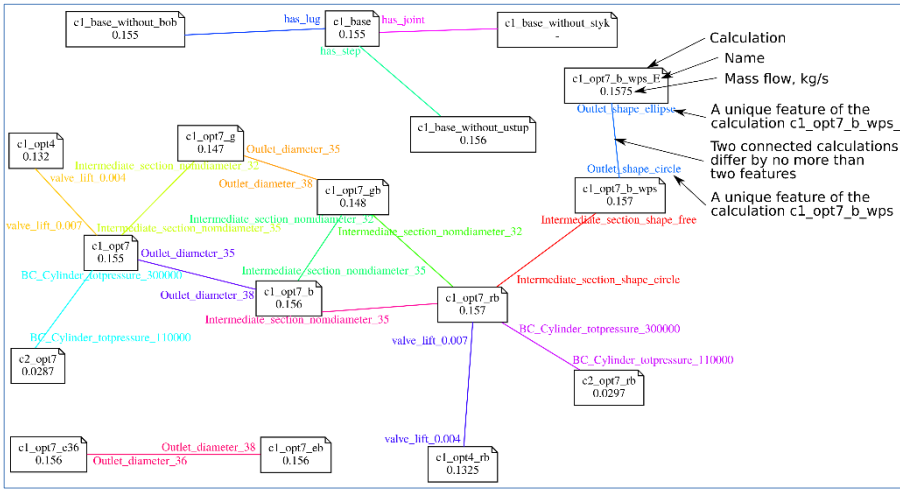

Fig. 6. An example of a proximity graph of calculations automatically generated from the knowledge base

Fig. 6 shows the proximity graph contracted by the program, where the connecting lines are drawn according to the criteria of “no more than two different points in the list of features”. The program displays these different points at the corresponding end of the connecting line. At the graph nodes, in addition to the calculation name, the value of the flow through the channel is displayed. Analysis of the graph by a user makes it easy to select ways to increase consumption, which is the main purpose of profiling, and to identify parameters that do not affect this value.

If it is necessary to detail the differences between the calculations for two objects not connected by the graph edges, this can be done using the command to compare two calculations. The screenshot is shown in fig. 7.

Name of calculation	Unique features	General features
c1_base	• has_lug • has_joint • has_step • Intermediate_section_shape_free • Intermediate_section_nomdiameter_34.8	• valve_lift_0.007 • Outlet_shape_circle • Outlet_nomdiameter_35 • BC_Cylinder_totpressure_300000 • with_turbulence_model_mu_const_0.01 • with_soft_NSF_2017 • for_substance_air_politrop • BC_Outlet_pressure_100000
c1_opt7	• Intermediate_section_shape_circle • Intermediate_section_nomdiameter_35	

Fig. 7. An example of a table comparing two calculations on lists of automatically generated features

Another typical question that a calculation specialist has when analyzing the results is the degree of influence of a parameter on the simulation results. The simplest way to answer this question is to conduct a separate experiment, despite the fact that the required information may already be implicitly contained in previous calculations (and sometimes even explicitly). Using a typical method of logging an experiment, it may be more difficult to reveal this information than to recalculate. Using the repository allows performing the appropriate analysis automatically. The corresponding script splits the bank of calculations into pairs, one of which contains the desired feature, and the other does not. The script sorts them in the order of proximity of two members in a couple to each other. An example of such analysis for this project is shown in fig. 8.

```
Feature analysis :Outlet_nomdiameter_35

Similarity      Calc with a feature    Calc without a feature
Similarity 2 :  c1_opt7                c1_opt7_b
Similarity 2 :  c1_opt7_g              c1_opt7_gb
Similarity 3 :  c1_opt4                c1_opt4_rb
Similarity 3 :  c1_opt7                c1_opt7_rb
Similarity 3 :  c1_opt7_g              c1_opt7_rb
Similarity 3 :  c2_opt7                c2_opt7_rb
Similarity 4 :  c1_opt4                c1_opt7_b
Similarity 4 :  c1_opt7                c1_opt7_gb
Similarity 4 :  c1_opt7_g              c1_opt7_b
Similarity 4 :  c2_opt7                c1_opt7_b

Shown 10 nearest pairs of 72
```

Fig. 8. The result of the script that analyzes the presence of a particular feature in the knowledge base.

As one can see, the representation of the bank of calculations in the knowledge base format provides ample opportunities for analyzing and visualizing calculations, which are impossible (or extremely time-consuming) when using simpler methods of modeling logging.

8. Conclusion

As the result, a software package has been created that implements the functioning of the repository for CFD calculations based on a semantic knowledge base. To ensure its correct operation, an ontology has been also created, which formulates the information model of calculation. A number of scripts have been written to support a user during the process of analyzing data in a database.

Currently, there is the developed prototype of this project and the first version (0.1) is released. All scripts are published under the GNU GPL Version 3 open source license on the bitbucket platform [13]. The author plans to work on the project to develop the user interface and provide more repository capabilities.

References

[1]. Junker U., Mailharro D. The Logic of ILOG (J) Configurator: Combining Constraint Programming with a Description Logic. In Proc. of the 18th International Joint Conference on Artifical Intelligence (IJCAI-03), Configuration Workshop, 2003, pp. 13-20.

[2]. A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. Artificial Intelligence. Vol. 152. Issue 2. 2004. pp. 213-234. DOI: 10.1016/S0004-3702(03)00117-6.

[3]. Blondet G., Le Duigou J., Boudaoud N. ODE: An Ontology for Numerical Design of Experiments. Procedia CIRP, vol. 50. 2016, pp. 496-501. 10.1016/j.procir.2016.04.199.

[4]. Ajit S., Sleeman D., Fowler D., Knott, D. Constraint capture and maintenance in engineering design. Artificial intelligence for engineering design analysis and manufacturing, vol. 22, 2008, pp 325-343. DOI:10.1017/S089006040800022X.

[5]. M. Ito, D. Ishihara, M. Tsuchiya, M. Otsuka, K. Tsuchimoto, Y. Miyazaki. Development of CAE Standardization System for Motorcycle Parts with Navigating Function for Designers. Honda R&D Technical Review, vol. 23, № 2. 2011. pp 90-96.

[6]. SHustova D.V., Borgest N.M. Development of the semantic bases of information systems in the design and manufacture of engineering products. Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh sistem [Open semantic technologies for the design of intelligent systems], №7, 2017, pp. 293-296. (in Russian)

[7]. Nazarov D.M., Borgest N.M. Managing a CFD model using a linked external database. XIII Korolyovskie chteniya. Mezhdunarodnaya molodyozhnaya nauchnaya konferentsiya, sbornik trudov [XIII Korolev readings. International Youth Scientific Conference, proceedings], 2015, p. 114. (in Russian)

[8]. CML-Bench™ – computer activity management system. Available at: http://fea.ru/article/cml-bench, accessed 27.08.2018. (in Russian)

[9]. ANSYS EKM: Simulation Data and Process Management Available at: https://www.ansys.com/products/platform/ansys-ekm, accessed 27.08.2018

[10]. Richard Cyganiak, David Wood, Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 25 February 2014. Available at: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225, accessed 27.08.2018

[11]. ISO 10303-1:1994 Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles.

[12]. Apache Jena - Apache Jena Fuseki. Available at: https://jena.apache.org/documentation/fuseki2/index.html, accessed 27.08.2018
[13]. Zenkin / repCAE – Bitbucket. Available at: https://bitbucket.org/zenkin/repcae, accessed 29.10.2018

Онтологический репозиторий для CFD-расчетов

В.А. Зенкин <vl.zenkin@gmail.com>
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
105005, Россия, г. Москва, ул. 2-я Бауманская, д.5. стр.1

Аннотация. На основе RDF-хранилища создан программный комплекс для формирования базы знаний, содержащей информацию о проведенных гидрогазодинамических расчетах. Комплекс представляет собой набор скриптов, написанных на языке bash и python, которые опубликованы под открытой лицензией GNU GPL 3. Инструмент предназначен для поддержки расчетчика при проведении поисковых исследований, не имеющих строгого, наперед заданного плана эксперимента или алгоритма решения задачи. Для формализации описания расчета, хранящегося в базе знаний, создана онтология, служащая его информационной моделью. В качестве вспомогательного средства для проведения автоматизированного сравнения расчетов друг с другом разработан механизм «компараторов» и «особенностей», также описанный в статье. Помимо систематизированного хранения данных комплекс обеспечивает возможность их автоматического и полуавтоматического анализа, в том числе представление банка расчетов в форме неориентированного графа, построение плоских и пространственных зависимостей, поиск сходных расчетов и т.п. В статье приводятся примеры обработки данных проекта по профилированию каналов в головке цилиндров поршневого двигателя.

Ключевые слова: методология CAE; онтология; управление знаниями; вычислительная газодинамика; семантические технологии

DOI: 10.15514/ISPRAS-2016-30(5)-15

Для цитирования: Зенкин В.А. Онтологический репозиторий для CFD-расчетов. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 249-264 (на английском языке). DOI: 10.15514/ISPRAS-2016-30(5)-15

Список литературы

[1]. Junker U., Mailharro D. The Logic of ILOG (J) Configurator: Combining Constraint Programming with a Description Logic. In Proc. of the 18th International Joint Conference on Artifical Intelligence (IJCAI-03), Configuration Workshop, 2003, pp. 13-20.

[2]. A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. Artificial Intelligence. Vol. 152. Issue 2. 2004. pp. 213-234. DOI: 10.1016/S0004-3702(03)00117-6.

[3]. Blondet G., Le Duigou J., Boudaoud N. ODE: An Ontology for Numerical Design of Experiments. Procedia CIRP, vol. 50. 2016, pp. 496-501. 10.1016/j.procir.2016.04.199.

[4]. Ajit S., Sleeman D., Fowler D., Knott, D. Constraint capture and maintenance in engineering design. Artificial intelligence for engineering design analysis and manufacturing, vol. 22, 2008, pp 325-343. DOI:10.1017/S089006040800022X.
[5]. M. Ito, D. Ishihara, M. Tsuchiya, M. Otsuka, K. Tsuchimoto, Y. Miyazaki. Development of CAE Standardization System for Motorcycle Parts with Navigating Function for Designers. Honda R&D Technical Review, vol. 23, № 2. 2011. pp 90-96.
[6]. Шустова Д.В., Боргест Н.М. Разработка семантических основ информационных систем при проектировании и производстве машиностроительных изделий. Открытые семантические технологии проектирования интеллектуальных систем, №7, 2017, стр. 293-296.
[7]. Назаров Д.М., Боргест Н.М. Управление CFD-моделью при помощи связанной внешней базы данных. XIII Королёвские чтения. Международная молодёжная научная конференция, сборник трудов, 2015, стр. 114.
[8]. CML-Bench™ – система управления деятельностью в области компьютерного инжиниринга URL: http://fea.ru/article/cml-bench. (Дата обращения 27.08.2018)
[9]. ANSYS EKM: Simulation Data and Process Management URL: https://www.ansys.com/products/platform/ansys-ekm. (Дата обращения 27.08.2018)
[10]. Richard Cyganiak, David Wood, Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225. (Дата обращения 27.08.2018)
[11]. Государственный стандарт Российской Федерации, ГОСТ Р ИСО 13030-1-99, "Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 1. Общие представления и основополагающие принципы". ИПК Издательство стандартов, 1999.
[12]. Apache Jena - Apache Jena Fuseki URL: https://jena.apache.org/documentation/fuseki2/index.html. (Дата обращения 27.08.2018)
[13]. Zenkin / repCAE - Bitbucket URL: https://bitbucket.org/zenkin/repcae. (Дата обращения 29.10.2018)

Проверка функциональных свойств смарт-контрактов методом символьной верификации модели

Е.С.Шишкин <evgeniy.shishkin@gmail.com>
ИнфоТеКС, Научный отдел
127287, Россия, Москва, Старый Петровско-Разумовский проезд, 1/23, стр.1, этаж 2

Аннотация. В статье рассматривается подход к проверке функциональных свойств смарт-контрактов платформы Ethereum методом символьной верификации модели. Описанный подход позволяет верифицировать выполнение 3х видов свойств на трассах ограниченной длины, а также осуществлять проверку выполнимости инварианта. Описана математическая модель среды исполнения смарт-контрактов, проведена формализация выделенных видов свойств в рамках этой модели, описана процедура трансляции всего перечисленного в язык ограничений SMT-решателя. Жизнеспособность предлагаемого подхода иллюстрируется на примере верификации макетного смарт-контракта MiniDAO, упрощённой версии известного TheDAO. За несколько секунд макет находит контр пример одному нетривиальному функциональному требованию, указывая на ошибку в бизнес-логике смарт-контракта. Насколько нам известно, эта работа - одна из первых попыток описать инструмент, помогающий осуществлять формальную проверку функциональных свойств смарт-контрактов в автоматическом режиме.

Ключевые слова: символьная верификация модели; смарт-контракты; блокчейн; формальная спецификация

DOI: 10.15514/ISPRAS-2018-30(5)-16

Для цитирования: Шишкин Е.С. Проверка функциональных свойств смарт-контрактов методом символьной верификации модели. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 265-288. DOI: 10.15514/ISPRAS-2018-30(5)-16

1. Введение

В конце 2008 г. Сатоши Накамото[1] опубликовал работу, в которой описал принципиальное устройство первой полностью децентрализованной платёжной системы под названием BitCoin [10].

Представляя собой распределенный реестр операций со счетами пользователей, система обладает уникальным сочетанием полезных свойств отказоустойчивости, неподменяемости вводимой информации, практической невозможностью цензурирования доступа, прозрачностью проводимых операций. До публикации Накамото, в мире и раньше использовали распределенные базы данных с повышенными гарантиями отказоустойчивости и аутентичности операций через использование ЭЦП, но все они опираются на наличие некоторой единой точки доверия — например администратора, который безраздельно обладает контролем над системой и сохраняемыми данными. В системе BitCoin, напротив, отсутствует доверенная сторона - пользователи доверяют исключительно описанному протоколу.

Семейство протоколов, подобных протоколу BitCoin, получило совокупное название протоколов типа блокчейн из-за их использования технологии зацепления блоков информации через вставку хэша предыдущего блок в следующий. Вскоре стало понятно, что технология блокчейн может быть использован не только как средство обмена ценностью через перевод криптовалюты, но и как распределенная вычислительная платформа по созданию надёжных отказоустойчивых нецензурируемых сервисов.

В 2015 г. состоялся первый релиз вычислительной платформы на базе блокчейн под названием Ethereum [18]. Платформа позволяет пользователям загружать в распределенную систему некую программу, описывающую желаемый бизнес-процесс. После публикации программы в блокчейне Ethereum, заинтересованные лица могут совершать вызовы в эту программу, меняя её внутреннее состояние, совершать криптовалютные переводы и, таким образом, взаимодействовать с остальными участниками процесса и программой. Программа в данном контексте называется смарт-контрактом, а пользовательские вызовы — транзакциями.

Неизменяемость смарт-контракта после публикации в блокчейне позволяет участникам бизнес-процесса «верить» в его «честность» - все возможные действия, а также уже совершенные транзакции программы видны любому желающему, поэтому не остаётся места различного рода манипуляциям с данными или бизнес-логикой. К сожалению, это же качество представляет и угрозу: если в логику смарт-контракта прокралась ошибка (случайная или специально привнесённая на этапе разработки), после записи контракта в блокчейн её невозможно исправить, а злоумышленник может воспользоваться уязвимостью в любой момент для извлечения выгоды.

[1] Это псевдоним. Настоящее имя автора до сих пор неизвестно.

Так, в 2016 г. из смарт-контракта TheDAO вывели объем криптовалюты, оцениваемый по биржевому курсу на момент инцидента в 60 млн. долларов [11]. Уязвимость была связана со спецификой поведения одной из программных конструкций языка программирования смарт-контрактов Solidity, на котором был записан TheDAO. С того момента имело место ещё несколько громких инцидентов [12] [13].

Необходимость строгой проверки смарт-контрактов на корректность на ранних стадиях разработки стала очевидна для разработчиков и заказчиков. Со временем стали появляться аудиторские компании, предоставляющие услугу ручного исследования смарт-контракта на соответствие заявленным требованиям. Зачастую аудит проводится неформально, «глазами», без строго обоснования сделанных заключений. Составляется отчёт о проведённом исследовании, где выдаются рекомендации по исправлению найденных ошибок, либо выносится заключение о надёжности смарт-контракта.

В этой связи стоит вспомнить, что смарт-контракт TheDAO также подвергался аудиту со стороны экспертов, включая самих создателей языка Solidity *и* специалиста по формальной верификации из Ethereum Foundation, однако это не помогло предотвратить печальных последствий! [14] Данный факт может служить аргументом к тезису о том, что, каков бы ни был уровень подготовки проверяющего эксперта, в его работе желательно присутствие инструмента *формальной* проверки программного артефакта. Люди склонны совершать ошибки.

Индустрии, на наш взгляд, требуется инструмент, который бы помогал проводить проверку соответствия смарт-контракта заявленной функциональной спецификации в автоматическом режиме.

В данной работе рассматривается возможность построения инструмента формальной верификации некоторых видов функциональных свойств смарт-контрактов. В качестве эталонной блокчейн платформы была выбрана платформа Ethereum, с языком программирования контрактов Solidity. Наличие эталонной платформы позволяет уже сейчас экспериментировать с реальными смарт-контрактами, при этом оставляя возможность перенести результаты работы на другую платформу, близкую по архитектуре.

Смарт-контракты в нашем случае записываются на языке программирования Sol - подмножестве языка Solidity, специально выделенном для упрощения процедуры проверки смарт-контракта, но достаточном для реализации нетривиальной бизнес-логики, в то время как функциональная спецификация задаётся либо в виде предикатов над состоянием контракта, либо в виде допустимых цепочек событий, испускаемых контрактом и взаимосвязях между этими событиями. Проверка осуществляется методом символьной верификации модели (symbolic model-checking). Модель извлекается из программы на языке Sol автоматически.

Вклад работы:

- Сформулировано несколько классов функциональных свойств, которые помогают описывать желаемое поведение смарт-контракта.
- Описан язык программирования смарт-контрактов Sol; язык выделен для целей формального анализа.
- Описана процедура кодирования программы Sol и спецификации в вид, пригодный для проверки SMT-решателем.
- Сделан макет, реализующий описанный метод на пример смарт-контракта MiniDAO - «младшего брата» смарт-контракта TheDAO. Проводится формальная проверка выполнимости нескольких функциональных требований. За несколько секунд макет находит контрпример, указывая на ошибку в логике MiniDAO.

Несмотря на то, что метод символьной верификации моделей программ — это хорошо известный способ проверки темпоральных свойств реагирующих систем, данная работа, насколько нам известно, представляет первую попытку применить этот метод к верификации функциональных свойств смарт-контрактов. Кроме этого, в литературе нам не попадалось работ, предлагающих способы специфицирования желаемого поведения смарт-контракта на формальном языке. Наша работа отчасти восполняет указанные пробелы. Результаты макетирования убеждают нас в том, что описанный подход является перспективным и заслуживает дальнейшей детальной проработки.

2. Смарт-контракты на платформе Ethereum

Объектом верификации в нашей работе являются смарт-контракты платформы Ethereum, записанные на подмножестве языка Solidity. Для краткости, мы опускаем описание работы платформы и языка программирования, ознакомиться с основами можно по материалам [18].

Язык Sol. Чтобы сделать процедуру символьной верификации программы смарт-контракта осуществимой на практике, мы выделили из языка Solidity некоторое подмножество, достаточно выразительное для осуществления интересных бизнес-сценариев, но не приводящее к астрономическому росту порождаемых состояний.

Для формирования представления о том, какие программные конструкции пользуются «спросом», в июле 2018 года было проведено небольшое исследование базы данных смарт-контрактов платформы Ethereum, снабжённых текстами программ на языке Solidity[2]. Из 27341 доступных программных текстов смарт-контрактов, только 23% используют хотя бы одну из форм циклов, и 26% используют динамические массивы. Между тем, эти программные конструкции сложнее всего подвергаются анализу, поэтому на данном этапе нашей работы было решено исключить их из рассмотрения.

[2]По базе https://etherscan.io/contractsVerified

В Solidity присутствует тип mapping который позволяет находить элемент без необходимости итерирования по коллекции. Этот тип частично снимает зависимость от циклов как средства организации вычисления. Следует вспомним также, что один из самых известных контрактов блокчейна Ethereum - TheDAO - не содержит циклов и рекурсии.

Отличия языка Sol от Solidity. 1) отсутствуют циклы for, do/while, рекурсия, взаимные вызовы функций; 2) отсутствуют механизмы динамического создания и удаления объектов через new и delete; 3) поддержка только статических массив; 4) ключевое слово var запрещено, все типы указываются явно; 5) в программе может быть единственный объект contract; 6) события не должны содержать более 4х аргументов; 7) тип address задаётся конечным множеством уникальных идентификаторов. В тексте программы запрещается напрямую указывать значение адреса; 8) нельзя вызывать методы других контрактов, а также динамически создавать экземпляры контрактов; 9) временно не поддерживаются операции со строками и битовые операции с целыми числами.

3. Модель системы

В нашей работе мы ограничимся системами, в которых существует единственный смарт-контракт, и в него производятся пользовательские вызовы. Такая аппроксимация подходит для большинства практических сценариев.

Введём такие обозначения: $\mathrm{N}_{256} \stackrel{\text{def}}{=} \{0 \dots 2^{256}-1\}$, $Addr \stackrel{\text{def}}{=} \{a_0 \dots a_n\}$, $\mathbb{B} = \{true, false\}$. Условимся, что каждой переменной состояния контракта присвоен уникальный постоянный номер. Обозначим множество всевозможных принимаемых значений переменных контракта как Val.

Пусть Ф означает множество публичных функций смарт-контракта, Е множество событий смарт-контракта.

Состояние всей системы зададим тройкой: $\sigma \stackrel{\text{def}}{=} \langle \sigma_c, b, t \rangle$ где σ_c - это состояние смарт-контракта, $b: Addr \to \mathrm{N}_{256}$ - балансы адресов системы, $t: \mathrm{N}_{256}$ - время последнего блока, относящегося к смарт-контракту. Мы обозначаем множество всевозможных состояний системы (не обязательно достижимых) через Σ, т.е. $\sigma \in \Sigma$.

Состояние смарт-контракта определим как $\sigma_c \stackrel{\text{def}}{=} \langle \sigma_{cs}, alive, eventlog \rangle$, где $\sigma_{cs}: \mathbb{N} \to Val$ задаёт текущее состояние переменных контракта (отображение уникального номера переменной в значение), $alive : \mathbb{B}$ - индикатор, указывающий является ли контракт активным, или он был удалён, $eventlog: \{\emptyset\} \cup E$-событие, сгенерированное смарт-контрактом в процессы выполнения транзакции, либо его отсутствие. Заметим, что в Solidity возможно генерировать сразу множество событий в процессе выполнения функции, и все они попадут в лог событий данной транзакции, но мы намеренно ограничиваемся только такими контрактами, в которых функция генерирует не более одного события. Это, с одной стороны, упрощает модель и процедуру проверки, с другой стороны *каждую конечную цепочку исходных событий можно при желании закодиро-*

вать одним событием. Поэтому налагаемое ограничение не является принципиальным.

Поясним природу переменной t. Транзакции в *Ethereum* выполняются узлами не по принципу FIFO, а путём группировки их в наборы, и последующей обработки сразу всего набора транзакций (последовательность выполнения транзакций из набора не определена). Эти наборы называются блоками. Каждому блоку в момент создания присваивается значение *blocktime* - время блока – момент времени, в который блок был создан. Это значение строго монотонно возрастает от блока к блоку и зачастую используется в бизнес-логике как источник относительного времени. Из программы смарт-контракта это значение можно прочитать вызовом функции *now*.

Поясним устройство множества Ф. В программе смарт-контракта определение функции состоит из названия функции, набора формальных аргументов, модификаторов, типа возвращаемого значения, и тела функции. Модификаторы могут задавать видимость (public, private), а также накладывать ограничение на возможность отправлять криптовалюту в смарт-контракт вместе с вызовом функции (payable). Принимая всё это во внимание, мы для каждой публичной функции смарт-контракта

$$functionf(arg_{0,}arg_{1,\dots},arg_n)public[payable][returns(T)]$$

ставим в соответствие функцию $f'(\sigma_i, v, s, t, p)$, где $\sigma_i \in \Sigma$ это состояние всей системы на момент выполнения функции, $v \in N_{256}$- количество криптовалюты, посылаемое вместе с вызовом, $s \in Addr$ - адрес отправителя транзакции,

$t \in N_{256}$ - время блока, в котором выполняется транзакция,

$p = (arg_0, arg_1, \dots, arg_n) \in \Pi$ - кортеж, состоящий из набора формальных параметров функции. Множество Ф задано набором таких функций.

Тело функции f' получается из тела функции f серией подстановок: вызов функции *now* заменяется на значение t, *msg.sender* заменяется на s, *msg.value* заменяется на v и т. д.

Функция f' возвращает изменённое состояние системы, которые мы обозначим за σ_{i+1}; само возвращаемое значение функции f на данный момент игнорируется - оно редко используется внешними пользователями для проверки результата выполнения функции, т.к. его неудобно отслеживать. Вместо этого чаще пользуются механизмом событий.

Введём несколько понятий, помогающих моделировать систему смарт-контракта во времени.

Определение. (Множество начальных состояний). Множество начальных состояний системы определяется как

$$I \stackrel{\text{def}}{=} \{\{\sigma_c^0, b_{0,} t_0\} \mid b_0 : Addr \to N_{256,} t_0 \in N_{256}\},\ \sigma_c^0 = \{\sigma_{cs}^0, true, \emptyset\},$$

здесь b_0 - функция, задающая балансы пользователей системы и смарт-контракта, t_0 время блока, в котором смарт-контракт был записан, σ_{cs}^0 состо-

яние переменных смарт-контракта сразу *после успешного вызова функции конструктора*.

В языке Sol мы запрещаем в конструкторе использовать любые выражения, способные привести к возникновению исключения, а также запрещается вызывать функции *transfer* и *selfdestruct*. Поэтому мы считаем, что конструктор всегда выполняется успешно и весь его побочный эффект заключается в присвоении значений переменным контракта.

Определение. (Трасса). Любая конечная последовательность состояний системы $\sigma = \sigma_0\sigma_1 \dots \sigma_{k-1}, \sigma_i \in \Sigma$ называется *трассой,* если справедливо $trace(\sigma) \stackrel{\text{def}}{=} \forall i \in \mathbb{N}, 0 \leq i < len(\sigma).\delta(\sigma_i, \sigma_{i+1})$, где $\delta(\sigma_i, \sigma_j)$ называется отношением шага (определено ниже), $\sigma_0 \in I$, и $len(\sigma)$ задаёт длину последовательности.

Множество всех возможных последовательностей состояний (не обязательно трасс) обозначим как 2^{Σ}.

В общем случае, из своего текущего состояния смарт-контракт может переходить в различные состояния, т.к. заранее неизвестно какую функцию и с какими параметрами пожелает вызвать тот или иной пользователь. Это закладывает недетерминизм в формировании любого последующего состояния трассы. Поэтому, когда мы рассуждаем про контракт, вместо одной трассы мы рассматриваем сразу множество всех возможных трасс. Только так мы можем гарантировать, что не упустим ошибочных состояний из внимания.

Определение. (Поведение). Пусть $\Sigma^{\star} \stackrel{\text{def}}{=} \{\sigma \mid \sigma \in 2^{\Sigma} \wedge trace(\sigma)\}$ - множество всех возможных трасс системы. Назовём это множество поведением системы.

Чтобы состояние σ_{i+1} «имело право» следовать за состоянием σ_iв последовательности трассы, пара (σ_i, σ_{i+1}) должна находится в определённом отношении, которое мы называем *отношением шага*. Для описания этого отношения, необходимо разобраться в природе транзакции.

Частично заданные функции. Функции $f' \in \Phi$ способны порождать исключения, т.е. прерывать выполнение функции с откатом всех ранее внесённых системных изменений, включая переводы криптовалюты, изменение значений переменных состояния контракта и т.д. Про такие функции мы говорим, что они заданы *частично*, т.е. определены не на всех значениях входных аргументов. Этот феномен, вероятно, можно было бы промоделировать, сделав отношение шага $\delta(\sigma_i, \sigma_j)$ рефлексивным. Но в этом случае, у нас бы появилось большое количество «мусорных» переходов, т.е. таких переходов, которые не ведут в новые состояния, и значит не представляют интереса в смысле проверки корректности поведения смарт-контракта. Чтобы отбросить такие переходы, мы вводим понятие предусловия для функций из Ф.

Определение. (Предусловие функции смарт-контракта). Назовём предусловием функции $f'(\sigma_i, v, s, t, p)$предикат $f_{pre}(\sigma, v, s, t, p)$ такой, что если он выполняется для заданных аргументов

$\sigma \in \Sigma,\ v \in N_{256},\ s \in Addr,\ t \in N_{256},\ p \in \Pi,$ то функция $f' \in \Phi$ определена на этих параметрах. Зададим множество $\Phi^{pre} \stackrel{\text{def}}{=} \{(f', f_{pre})\}$, т.е. каждую функцию $f' \in \Phi$ снабдим её предусловием f_{pre}.

Определение. (Отношение шага). Из состояния σ_iвозможно сделать шаг в состояние σ_{i+1}если существует набор v, s, t, p такой, что хотя бы одна функция $f' \in \Phi$ определена на значениях σ_i, v, s, t, p.

Множество всех таких пар состояний мы называем отношением шага:

$$\Delta \stackrel{\text{def}}{=} \{(\sigma_i, \sigma_j) : \exists (f', f_{pre}) \in \Phi^{pre}, v, s, t, p.\, \sigma_j = f'(\sigma_i, v, s, t, p) \wedge f_{pre}(\sigma_i, v, s, t, p)\}$$

Принадлежность к этому множеству определяется предикатом

$$\delta(\sigma_i, \sigma_j) \stackrel{\text{def}}{=} ((\sigma_i, \sigma_j) \in \Delta)$$

4. Задача верификации

Ранее была описана модель системы взаимодействия пользователей со смарт-контрактом. Сформулируем решаемую задачу проверки функциональных свойств смарт-контракта, заданных на языке спецификации.

Определение. (Задача верификации). Пусть P - предикат над трассой. Нужно установить, что $\forall \sigma \in \Sigma^{\star}.\, \sigma \vDash P$, т.е. что поведение смарт-контракта удовлетворяет P. Предикат P в этом контексте называем (формальной) спецификацией на смарт-контракт.

В зависимости от вида свойства (свойство конкретных состояний или событийное свойство), предикат P принимает либо одно состояние $P : \Sigma \to \mathbb{B}$ либо цепочку длины не более k $P : \Sigma_k^{\star} \to \mathbb{B}$.

Здесь и далее $\Sigma_k^{\star} = \{\sigma \in \Sigma^{\star} \mid len(\sigma) \leq k\}$, где $len(\sigma)$ задаёт длину последовательности.

Сформулируем виды функциональных свойств, которые мы хотим уметь проверять, в терминах изложенной ранее модели.

Определение. (Инвариант). Предикат $P : \Sigma \to \mathbb{B}$ называется инвариантом системы, если

$$\forall \sigma_0 \in I.\, P\ \sigma_0 \wedge \forall \sigma_i, \sigma_j \in \Sigma. \left(\delta(\sigma_i, \sigma_j) \wedge P\ \sigma_i\right) \to\ P\ \sigma_j$$

Определение. (Свойство на трассах длины k).

Предикат $P : \Sigma \to \mathbb{B}$ называется свойством трассы длины k, если

$$\forall \sigma^{\star} \in \Sigma_k^{\star}, i \in \mathbb{N}.\, i \leq len(\sigma^{\star}) \to P(\sigma_i^{\star})$$

Определение. (Паттерн возникновения событий). Свойства этого класса задаются предикатами над трассами длины не более k , т.е. $P : \Sigma_k^{\star} \to \mathbb{B}$. Рассмотрим одно из них, остальные определяются по аналогии. Если произошло событие $E_1(p_0, \ldots, p_n)$, после которого в какой-то момент происходит событие $E_2(m_0, \ldots, m_k)$, то между ними не должно возникнуть событие $E_3(n_0, \ldots, n_m)$, т.е.

$$P(\sigma^{\star}) = \exists i,j \in N.\, \sigma_i^{\star}[eventlog] = E_1 \wedge \sigma_j^{\star}[eventlog] = E_2 \wedge i < j \leq len(\sigma_i) \rightarrow$$
$$\forall k \in N, i < k < j.\, \sigma_k^{\star}[eventlog] \neq E_3$$

Если есть дополнительные зависимости между параметрами событий $p_0,\ldots,p_n,m_0,\ldots,m_k,n_0,\ldots,n_m$, то они добавляются к указанному предикату.

Определение. (Возможность выполнения транзакции). Свойства этого класса задаются предикатом над трассами длины не более k, т.е. $P{:}\,\Sigma_k^{\star} \rightarrow \mathbb{B}$. Рассмотрим одно из них, остальные определяются по аналогии. Если произошло событие $E_1(p_0,\ldots,p_n)$, после которого в какой-то момент происходит событие $E_2(m_0,\ldots,m_q)$, то между ними всегда возможно успешно выполнить $f'(\sigma,v,s,t,p)$.

$$P(\sigma^{\star}) = \forall i,j \in N.\ \sigma_i^{\star}[eventlog] = E_1(p_{0,}\ldots,p_n) \wedge$$
$$\sigma_j^{\star}[eventlog] = E_2(m_{0,}\ldots,m_q) \wedge$$
$$i < j \leq len(\sigma^{\star}) \rightarrow \forall k \in N, i < k < j.\, f_{pre}(\sigma_k^{\star},v,s,t,p)$$

Если есть дополнительные зависимости между параметрами событий $p_0,\ldots,p_n,m_0,\ldots,m_q$и параметрами вызываемой функции σ,v,s,t,p , то они добавляются к указанному предикату.

5. Конструирование модели смарт-контракта

Построить модель смарт-контракта и спецификации означает задать такой набор объектов: $(\Phi^{pre},E,Addr,I,k,\Sigma_k^{\star},P)$, где k означает максимальную длину анализируемой трассы. Обсудим процедуру построения каждого из этих объектов для какого-то заданного смарт-контракта.

Множество $Addr$. В системе Ethereum множество *Addr* совпадает с множеством $\{0\ldots 2^{160}-1\}$, но для целей символьной верификации такое множество оказывается слишком большим. Дело в том, что его размер влияет на количество достижимых состояний: параметр s в функции $f'(\sigma_i,v,s,t,p)$ выбирается из множества *Addr*, а значит чем оно крупнее, тем больше возможностей выбора. Поэтому, в верифицируемых моделях множество адресов $\{a_0\ldots a_n\}$ мы стараемся выбирать как можно меньшего размера, но такого, чтобы было возможно пройти по всем принципиальным сценариям выполнения. Оптимальный размер этого множества можно определить только исходя из понимания бизнес логики конкретного контракта и на данный момент никак не автоматизировано. По умолчанию, полагаем $Addr = \{noAddr, addr_0, addr_1, addr_2, contractAddr\}$.

Элемент *noAddr* соответствует отсутствию адреса - то, что в коде обычно обозначается как address(0). Элемент *contractAddr* задаёт адрес анализируемого контракта.

Множество Φ^{pre}. Строится автоматически: для каждой *публичной* функции f' смарт-контракта (кроме конструктора) строится предикат f_{pre} над набором

переменных σ, v, s, t, p. Это делается методом символьного исполнения кода функции. В местах кода, где происходит вызов другой функции, мы производим встраивание тела вызываемой функции.

Потенциально опасные конструкции. Конструкции, способные привести к возникновению исключения, на данный момент: операция деления целых чисел, взятие остатка от деления; функции mulmod, addmod; отправка криптовалюты через transfer; функции assert, require**,** revert; оператор throw; вызов нерayable функции с параметром $v > 0$; совпадение адреса s с адресом смарт-контракта *contractAddr;* попытка вызова функции смарт-контракта, который был удалён, т.е. $\sigma_{cs}[alive] = false$

Этот список будет расширяться впоследствии. Так как в языке Sol нет циклов и рекурсии (инструмент верификации проводит синтаксическую проверку программы перед построением модели), то символьное исполнение кода функции гарантировано завершается, и набор предикатов f_{pre} будет получен за конечное время.

Множество E. Строится автоматически из списка определённых в смарт-контракте событий. События, которые не используются ни в одной из функций, отбрасываются.

Множество I. Множество начальных состояний смарт-контракта.

Значение k. Максимальная длина анализируемой трассы. Задаётся в явном виде пользователем. Если проверяется инвариант, этот параметр игнорируется.

Множество $\Sigma_k^\star$. Множество задаётся неявно, через построение системы ограничений на наборе состояний $\sigma_{[0..k-1]}$и наборе параметров (v, s, t, p), где i-й набор соответствует параметрам, передаваемым в i-й по счёту вызов одной из функций смарт-контракта.

Определим отношение перехода:

$$transition(\sigma_i, \sigma_{i+1}) = \bigvee_{0 \le j < n} (f_j^{pre}(\sigma_i, v_i, s_i, t_i, p_i) \wedge \sigma_{i+1} = f_j'(\sigma_i, v_i, s_i, t_i, p_i)),$$

где $n = |\Phi^{pre}|$

Определим путь между состояниями: $path(\sigma_{[0..k]}) = \bigwedge_{0 \le i < k} transition(\sigma_i, \sigma_{i+1})$

Путь отличен от трассы тем, что первое состояние в последовательности не обязано быть начальным. Путь длины 0 содержит единственное состояние, в нём не совершается ни единого перехода.

Введём дополнительно требование на монотонность времени $T = \bigcup_{i=0..k-1} \{t_i < t_{i+1}\}$, требование на начальные состояния: $I(\sigma_0)$, требование невозможности вызова функций смарт-контракта с адреса самого смарт-контракта: $NoSelfCall = \bigcup_{i \in \{0..k-1\}} \{s_i \neq contractAddr\}$, требование невозможности вызова функций смарт-контракта с адреса *noAddr*:

$$NoAddrCall = \bigcup_{i \in \{0..k-1\}} \{s_i \neq noAddr\}$$

В этом случае, кат $I(\sigma_0) \wedge T \wedge NoSelfCall \wedge NoAddrCall \wedge path(\sigma_{[0..k-1]})$ описывает ограничения, выполнение которых в контексте SMT-решателя задаёт присваивания переменным $\sigma_{[0..k-1]}$ и (v, s, t, p), неявно «генерируя» множество $\Sigma_k^\star$.

Предикат P. Предикат формируется путём трансляции функционального свойства в предикат первого порядка, как это было описано в разделе.

Таким образом, из программы на языке Sol возможно автоматически извлечь модель, пригодную для передачи в SMT-решатель. Единственное, что требуется от пользователя, это указать функциональную спецификацию *P*, длину трассы *k*, количество элементов в множестве *Addr*.

6. Алгоритм проверки спецификаций

В этом разделе мы описываем устройство алгоритмов проверки функциональных свойств. Пусть $P(\sigma)$ задаёт проверяемое свойство. Выражение $SAT(e, Vars)$ означает, что утверждение *e* проверяется SAT/SMT решателем на выполнимость, т.е. ищется такой набор присваиваний для переменных из *Vars* такой, что вся логическая формула *e* становится истиной. Если такое присваивание удаётся найти, то функция возвращает *true*. Иначе возвращает *false*. Результат *unknown* не рассматривается, т.к. мы находимся в рамках полностью разрешимых теорий для которых этот результат означает нехватку выделенного времени на поиск решения. Предикат $path(\sigma_{[0..n]})$ определяется, как было указано ранее. Каждый из алгоритмов возвращает *true*, если свойство $P(\sigma)$ выполняется, иначе *false*. В используемом псевдокоде выражение $Vars = \{p_i : t_i\}$ означает, что для каждого p в набор пропозиционных переменных решателя добавляется переменная типа *t*, либо набор однотипных переменных в случае массива.

Некоторые из этих алгоритмов уже публиковались ранее. Так, алгоритм проверки выполнимости свойства на пути длины k подробно рассмотрен в различных вариациях в [20], а алгоритм проверки инварианта хорошо известен. Тем не менее, чтобы сделать текст самодостаточным, мы приводим псевдокод всех используемых нами верифицирующих алгоритмов в одном месте.

Алгоритм 1. Проверка выполнимости инварианта

```
1: Vars = {σ[0,1]:Σ, v[0,1]:N256, s[0,1]:Addr, t[0,1]:N256, p[0,1]:Π}
2: if (SAT (I(σ0) ∧ ¬P(σ0), Vars)) {
3:     print s0
4:     return false
5: }
6: if (SAT (P(σ0) ∧ δ(σ0,σ1) ∧ ¬P(σ1), Vars)) {
7:     print fpre, σ0, σ1
8:     return false
9: }
10:return true
```

Обоснование алгоритма 1. В строке 2 мы проверяем выполнимость $P(\sigma)$ во всех начальных состояниях. Если проверка в строке 2 прошла успешно, мы переходим к проверке индуктивного шага: предполагая, что $P(\sigma)$ выполняется в каком-либо σ_0 (не обязательно начальном) и из него можно перейти в другое состояние σ_1, то $P(\sigma)$ будет выполняться и в σ_1.

Алгоритм 2. Проверка выполнимости свойства на пути длины k

```
1: Vars = {σ[0..k-1]:Σ, v[0..k-1]:N256, s[0..k-1]:Addr, t[0..k-1]:N256,
           p[0..k-1]:Π}
2: i = 0
3: while (i < k) do {
4:   if (SAT (I(σ0) ∧ path(σ[0..i]) ∧ ¬P(σi), Vars)) {
5:             print σ[0..i]
6:             return false
7:   }
8:   i = i + 1
9: }
10:return true
```

Обоснование алгоритма 2. Мы хотим убедиться, что для любой трассы *длины не более k* свойство $P(\sigma)$ будет выполняться во всех состояниях этой трассы. Для проверки этой гипотезы, мы последовательно, начиная с $i = 0$ (в этом случае мы проверяем отдельные точки – начальные состояния), просим решатель найти хотя бы один пример, в котором бы гипотеза нарушалась, и делаем так вплоть до $i = k - 1$, после чего алгоритм завершается.

Алгоритм 3. Проверка паттерна возникновения событий

```
1: Vars = {σ[0..k-1]:Σ, v[0..k-1]:N256, s[0..k-1]:Addr,
           t[0..k-1]:N256, p[0..k-1]:Π, m,n,q:N256}
2: i = 3
3: while (i < k) do {
4:     if (SAT (I(σ0) ∧ path(σ[0..i]) ∧  σ^cs_m[eventlog] = E1 ∧
            σ^cs_n[eventlog] = E2 ∧ (m < n) ∧ (q > m) ∧ (q < n) ∧
            σ^cs_q[eventlog] = E3, Vars) {
5:           print σ[0..i]
6:           return false
7:     }
8:     i = i + 1
9:}
10:return true
```

Обоснование алгоритма 3. Мы хотим убедиться, что любая трасса *длины не более k* удовлетворяет условию: если в трассе возникло событие E_1, после которого возникло E_2, то между этими событиями не возникает E_3 (гипотеза). Это условие задаёт паттерн возникновения событий в трассах смарт-контракта, и могло бы быть записано на языке регулярных выражений: $.* E_1(\neg E_3) * E_2.*$, при ограничении длины входных строк параметром k.
Проверка этого свойства делается путём последовательного, начиная с $i = 3$, поиска примера, опровергающего гипотезу. Поиск прекращается после того, как все трассы длины до k были проверены (условие в строке 3).

Алгоритм 4. Проверка возможности осуществления вызова функции

```
1: Vars = {σ[0..k-1]:Σ, v[0..k-1]:N256, s[0..k-1]:Addr, t[0..k-1]:N256,
                    p[0..k-1]:Π, m,n,q:N256}
2: i = 3
3: while (i < k) do {
4:     if (SAT (I(σ0) ∧ path(σ[0..i]) ∧  σ^cs_m[eventlog] = E1 ∧
            σ^cs_n[eventlog] = E2 ∧ (m < n) ∧ (q > m) ∧
            (q < n) ∧ ¬f_pre(σq, vq, sq, tq, pq), Vars))  {
5:                 print σ[0..i], vq, sq, tq, pq
6:                 return false
7:     }
8:     i = i + 1
9: }
10:return true
```

Обоснование алгоритма 4. Мы хотим убедиться, что любая трасса *длины не более k* удовлетворяет условию: если в трассе возникло событие E_1, после которого возникло E_2, то строго между этими событиями во всех состояниях возможно успешно выполнить функцию смарт-контракта $f(\sigma, v, s, t, p)$. Успешность вызова функции с заданными параметрами описывается выполнением предиката $f_{pre}(\sigma, v, s, t, p)$ (гипотеза).

Проверка этого свойства делается путём последовательного, начиная с $i = 3$, поиска примера, опровергающего гипотезу, то есть примера такой трассы, чтобы строго между E_1 и E_2 в каком-то из состояний предикат $f_{pre}(\sigma, v, s, t, p)$ не выполнялся. Взаимное расположение событий друг относительно друга задаётся с помощью отношений номеров состояний трассы, в которых соответствующие события возникли. Поиск прекращается после того, как все трассы длины до k были проверены (условие в строке 3). Трассы длины меньше 4х элементов слишком короткие, чтобы их проверять.

7. Описание макетного образца

В целях апробирования описанной методики, мы запрограммировали смарт-контракт MiniDAO - упрощённый вариант смарт-контракта TheDAO, и попробовали отыскать контр пример, демонстрирующий нарушение описанного в спецификации требования.

Программа смарт-контракта MiniDAO записана на языке Sol, а проверяемые свойства закодированы в виде соответствующих предикатов. Мы оттранслировали вручную оба артефакта в язык ограничений SMT-решателя и провели поиск контр примеров, фиксируя длительность проверки разного типа свойств с различными значениями параметров модели.

Мы кратко опишем логику работы MiniDAO и сформулируем несколько утверждений, которые будут служить частичной спецификацией на этот контракт.

7.1 Смарт-контракт MiniDAO

MiniDAO - это смарт-контракт, реализующий возможность привлечения криптовалютных инвестиций в новый проект. В смарт-контракте предусмотрено два вида участников: инвестор и подрядчик. Инвестор - это тот, кто вносит средства в фонд смарт-контракта, и далее голосует «за» либо «против» поддержки предложенного кем-то проекта. Подрядчик - это сторона, которая предлагает новый проект и занимается его реализацией, возвращая сумму инвестиций и дивиденды инвесторам через смарт-контракт. Интерфейс смарт-

```
interface ERC20Interface { /* Стандартный интерфейс ERC20 */ }
interface MiniDAOInterface {
    function deposit() public payable;
    function vote(uint proposalId, bool supportsProposal)
public;
    function refund() public;
    function propose(address recepient, uint amount,
                     string text) public;
    function execute_proposal() public;
    event Voted(address voter, uint proposalID,
                bool supportsProposal);
    event Refunded(address investor, uint tokens);
    event Deposited(address investor, uint tokens);
    event ProposalAdded(uint amount, uint proposalID);
    event ProposalExecuted(uint proposalID);
    event ProposalRejected(uint proposalID);
}
contract MiniDAO is MiniDAOInterface, ERC20Interface { ... }
```

контракта приведён на рис.1. Полный текст смарт-контракта доступен по ссылке[3].

.Рис. 1. Интерфейс смарт-контракта MiniDAO
Fig. 1. MiniDAO smart-contract interface

Чтобы дать инвестору возможность вывести свои средства из смарт-контракта MiniDAO, реализован метод refund(). Возврат осуществится только в том случае, если инвестор не голосовал за заявку. Смарт-контракт отправляет на адрес инвестора количество эфира пропорционально количеству токенов на внутреннем балансе инвестора, т.е. ровно столько, сколько инвестор вложил в фонд MiniDAO.

Инвесторы могут переводить на счета других инвесторов свои токены miniDAO. Это реализуется через поддержку стандартного интерфейса токенов ERC20.

Предположим, мы заинтересованы привлечь как можно больше инвесторов в наш фонд miniDAO. Чтобы минимизировать страх инвестора потерять свои деньги, мы заявляем, что наш смарт-контракт обладает таким функциональным свойством: «*Если вы не проголосовали ни за одно инвестиционное предложение, то вы всегда сможете забрать свои средства обратно*». В качестве аргумента мы указываем на программный код функции refund, которая отвечает за возврат средств инвестору.

```
  function refund() public {
        address sender = msg.sender;
        uint tokens = balance[sender];
        require (isVoted[0][sender] ==
false);
        require (tokens > 0);
        require (DAO_tokens_emitted >=
tokens);
        DAO_tokens_emitted -= tokens;
        balance[sender] = 0;
        sender.transfer(tokens *
DAO_token_price);
        emit Refunded(sender, tokens);
    }
```

Функция выглядит просто и убедительно. Однако свойство, тем не менее, не выполняется.

Атака большинства. Рассмотрим сценарий возможной атаки. Два инвестора вложили в инвест фонд MiniDAO криптовалюты суммой на X и Y токенов соответственно. Предположим, что появляется третий инвестор, который вкладывает криптовалюты объёмом на 2∗ (X+Y) токенов. У этого инвестора получается большинство голосов (2/3≈66%) при принятии решения о заявке.

[3]https://bitbucket.org/unboxed_type/minidao/src/master/contracts/MiniDAO.sol

Так как инвестору не запрещено быть подрядчиком, то этот инвестор регистрирует собственную заявку с указанием своего адреса и необходимую сумму в размере 3∗ (X+Y) токенов. Далее, из-за того, что его голос - решающий, он голосует за собственную же заявку и после этого вызывает *execute_proposal*. Все средства из фонда MiniDAO перейдут на счёт этого инвестора-злоумышленника, включая те средства, которые внесли первые два инвестора. Получается очевидное нарушение функционального требования: первый и второй инвестор не голосовали, но свои деньги они уже точно не вернут.

Атака большинства (в несколько другой форме) описана в оригинальной работе TheDAO [14]. *Предположим, что мы не знаем про атаку большинства и хотим попросить верифицирующий инструмент проверить, выполняется ли заявленное функциональное свойство.*

7.2 Функциональные свойства MiniDAO

В качестве примера сформулируем три свойства, которые мы хотели бы проверить с помощью инструмента верификации.

Свойство DepositedNotVotedRefund. Если инвестор с адресом inv вносил депозит, но при этом ни разу не голосовал за какую-либо заявку, он всегда сможет вернуть свои средства путём вызова функции refund.

$$DepositNotVotedRefund(\sigma) \stackrel{\text{def}}{=} \exists i, inv, s, 1 \le i < k.$$
$$\sigma_{cs}^{i}[logs] = Deposited(inv, s) \ \wedge \ \exists j, inv_1, s_1, id, 1 \le j < k.$$
$$\sigma_{cs}^{j}[logs] = ProposalAdded(inv_1, s_1, id) \wedge \forall n, id_1, 1 \le n < k.$$
$$\sigma_{cs}^{n}[logs] \ne Voted(inv, id_1, True) \wedge \sigma_{cs}^{n}[logs] \ne Voted(inv, id_1, False) \wedge$$
$$\sigma_{cs}^{n}[logs] \ne Refund(inv) \wedge \sigma_{cs}^{n}[logs] \ne Transfer(inv) \rightarrow$$
$$\forall m, i \le m < k. \exists v, t, p. refund_{pre}(\sigma_m, v, inv, t, p)$$

В данном случае мы используем язык логики первого порядка, так как его легче всего оттранслировать в язык ограничений SMT-решателя.

Свойство InvDaoBalanceEquTokens. В любом достижимом состоянии системы, сумма остатков токенов miniDAO на балансах пользователей всегда должна быть равна количеству эмитированной криптовалюты, т.е. для всех достижимых состояний системы,

$$InvDaoBalanceEquTokens(\sigma_k)$$
$$\stackrel{\text{def}}{=} \sum_{i \in Addr} \sigma_{cs}^{k}[daoBalance[i]] = daoTokensEmitted$$

Свойство RejectedNotExecuted. Инвестиционное предложение, которые было отклонено, не может получить перечисление криптовалюты из смарт-

контракта.

$$RejectedNotExecuted(\sigma) \stackrel{\text{def}}{=} \forall n. \exists i, j \in N.$$
$$\sigma_{cs}^{i}[logs] = ProposalAdded(n, amount) \wedge \sigma_{cs}^{j}[logs] = ProposalRejected(n) \wedge$$
$$i < j \rightarrow \forall k, i < k < j. \sigma_{cs}^{k}[logs] \neq ProposalExecuted(n)$$

7.3 Поиск ошибок в контракте MiniDAO

Мы проверяем смарт-контракт методом символьной верификации модели. Согласно введённому ранее определению, задать модель означает задать набор $(\Phi^{pre}, E, Addr, I, k, \Sigma_k^{\star}, P)$, после чего мы можем выполнить один из алгоритмов проверки функционального свойства.

Разрабатываемый инструмент будет строить указанные объекты автоматически, по исходному коду контракта, за исключением задания количества участников *Addr*, длины трассы *k* и проверяемого свойства *P*.

Так как инструмент проверки находится в стадии разработки, мы провели построение указанных объектов вручную, получив на выходе модель системы исполнения смарт-контракта и проверяемого функционального свойства, закодированную на языке SMT-решателя. В этой работе мы использовали SMT-

0. NoEvent
1. Deposited, sender = addr4, tokens = 2976
2. Deposited, sender = addr2, tokens = 1672
3. ProposalAdded, sender = addr4, amount = 4648, proposalID = 1
4. Voted, sender = addr4, proposalID = 1, supports=1
5. ProposalExecuted, proposalID = 1

step = 5, investor = addr2

решатель Z3 компании Microsoft [16].

Рис. 2. Трасса, опровергающая функциональное свойство DepositedNotVotedRefund.
Fig. 2. The counter-example for DepositedNotVotedRefund property.

После трансляции исходного кода смарт-контракта MiniDAO в представление SMT-решателя и проведения нескольких оптимизаций, была получена модель, проверка которой за несколько секунд (см. таблицу) синтезировала контр примеры, указывая на ошибку в бизнес-логике контракта.

Так, проверка свойства *DepositedNotVotedRefund* выявила контр пример, представленный на Рис.2. Первое событие NoEvent означает начальное состояние смарт-контракта. Указан набор событий, который ведёт к ошибочному состоянию. Инвестору addr4 не удаётся вызвать refund после события 5.

Проверка свойства *InvDaoBalanceEquTokens* также выявила ошибку. В первоначальной логике смарт-контракта MiniDAO, метод vote обнулял количество токенов на счёту инвестора, выполняя присваивание: daoBalance[msg.sender] = 0. Ошибка была устранена, после чего свойство прошло проверку.

Проверка свойства *RejectedNotExeceted* прошла успешно: не было обнаружено ни одной трассы, опровергающей сформулированное свойство.

Оптимизация модели. Известно, что SAT/SMT-решатели весьма чувствительны к изменению параметров проверяемой системы. После первоначальной трансляции исходного кода смарт-контракта MiniDAO в представление SMT-решателя, без оптимизаций, нам не удалось добиться от решателя результата за приемлемое время, поэтому было решено провести оптимизацию модели.

Оптимизацией мы называем уменьшение мощности множества возможных значений различных параметров системы (σ_i, v, s, t, p). Так, например, вместо множества N_{256} для v и t было положено множество N_{16}. Ещё одним оптимизирующим приёмом можно считать выбор начальной длины трассы, с которой начинается анализ. Очевидно, что при неудачном выборе размеров множеств и начальной длины трассы, есть вероятность пропустить сценарии, ведущие к ошибке.

8. Результаты работы макета

Изменяя размеры множеств, из которых выбираются значения состояний модели, а также длину пути, была составлена таблица с результатами измерений времени работы решателя на соответствующей модели, см. табл. 1.

Следует иметь ввиду, что эти результаты могут давать лишь *приблизительное представление* о том, в каких пределах лежит время поиска контр примера в зависимости от параметров модели: алгоритм, на который опирается SAT/SMT-решатель чувствителен к любым изменениям в модели, а в некоторых сценариях может на одной и той же модели с одинаковыми параметрами выдавать разные результаты, по скорости и контр примеру (недетерминизм).

Эксперименты проводились на машине Intel Core i7-4770, 4 ядра 3.4 GHz , 32 GB RAM под управлением ОС Linux Fedora 28 x64, запущенная в виртуальной машине VirtualBox 5.1.24, под управлением ОС Windows 7. Использовался SMT-решатель Z3 версии 4.7.1, 64 bit, модель записана с использованием расширения Z3Py для Python.

9. Обзор схожих работ

Исследованию различных методов поиска и предотвращения уязвимостей в смарт-контрактах посвящено множество работ, среди прочих [1] [2] [3] [4] [5] [6] [7] [8].

В работе [2] авторы приводят исчерпывающий список известных уязвимостей языка Solidity и виртуальной машины EVM, детально описывается механизм известной атаки на контракт TheDAO. Работа не стремится предложить какие-либо решения для верификации смарт-контрактов, но отлично освещает все известные на момент написания уязвимости языка и платформы.

Табл. 1. Левая таблица - длительность проверки свойства DepositedNotVotedRefund. Результат >N означает, что мы прервали работу решателя по истечении N секунд. Средняя таблица - длительность проверки свойства RejectedNotExeceted. Правая таблица - длительность проверки свойства InvDaoBalanceEquTokens.

Table 1. Left section: a duration of checking DepositedNotVotedRefund property. Abbr. «> N» denotes the fact that we have stopped the SMT solver after N seconds. Middle section: a duration of checking RejectedNotExecuted property. Right section: a duration of checking InvDaoBalanceEquTokens property.

Начальная длина трассы	Ширина целого числа, в битах	Время проверки, сек
6	16	34
8	16	7
12	16	474
6	32	9
8	32	115
12	32	> 660

Начальная длина трассы	Ширина целого числа, в битах	Время проверки, сек
6	16	12.9
8	16	9.4
6	32	46
8	32	30

Ширина целого числа, в битах	Время проверки, сек
16	243
24	997

Работа [3] предлагает способ статического анализа смарт-контрактов, записанных на языке Solidity* (упрощённая версия Solidity без циклов), с помощью трансляции в язык F* вместе со специально введёнными типами (монада Eth), помогающими отслеживать отсутствие должной обработки ошибок после вызова функций, а также обработке результата функции send. Предложенный в статье подход нацелен на устранение типовых ошибок в реализации смарт-контрактов на языке Solidity, при этом высокоуровневые функциональные свойства никак не проверяются.

В [1] исследуется возможность программирования смарт-контрактов на функциональном языке Idris; используя выразительную силу системы типов данного языка, авторы вводят несколько алгебраических типов, помогающих устранить определённый класс операционных ошибок на этапе компиляции. Более того, представлен backend для компилятора Idris, транслирующий код контрактов в исполняемый EVM байткод. Аналогично, в работе никак не адресован вопрос проверки высокоуровневых функциональных свойств смарт-контракта.

Работы [4], [7] описывают инструменты статического анализа смарт-контрактов, основанных на символьном исполнении программы смарт-контракта с поиском условий, выполнение которых влечёт переход управления на потенциально опасные участки кода, а также поиску в исходном коде смарт-контракта паттернов некоторых типовых уязвимостей.

В работе [6] описывается инструмент статического анализа смарт-контрактов ZEUS, способный искать нарушение заданных пользователем политик. Политики описываются как утверждения о состоянии переменных контракта с поддержкой арифметики (пропозиционные высказывания с арифметикой). Кроме этого, ZEUS ищет типичные уязвимости в программах на языке Solidity. В отличие от нашей работы, ZEUS не способен анализировать логику работы контракта, растянутую во времени, а нацелен исключительно на свойства безопасности достижимых состояний (safety).

Пожалуй, первой попыткой использовать интерактивную среду построения доказательств для проверки корректности смарт-контракта является работа [5]. Автор закодировал семантику инструкций EVM сначала в среде Coq, а затем и в Isabelle, что позволяет верифицировать свойства смарт-контрактов на уровне байт-кода EVM. Предоставляя наивысшие гарантии на корректность проверенного артефакта, подход обладает недостатками: пользователь должен иметь специальную квалификации и сам процесс построения доказательств может занять длительное время.

В работе [8], автор формализует часть языка Solidity, описывая операционную семантику некоторых конструкций. Формализация ведётся в среде построения доказательств Coq.

10. Заключение

В статье описан подход к проверке некоторых видов функциональных свойств подмножества языка Solidity методом символьной верификации модели. Описана модель исполнения смарт-контракта, позволяющая проверять функциональные свойства, заданные 4-мя возможными способами. Впервые представлен способ описания поведения смарт-контракта, заданный цепочкой генерируемых событий.

На примере смарт-контракта MiniDAO показана практическая применимость описываемого подхода к верификации. В качестве дальнейших работ по данной теме мы хотели бы особенно выделить такие направления:

1. Описание формального языка спецификации поведения смарт-контракта, основанный на событиях, генерируемых смарт-контрактом. Сейчас такие свойства записываются на языке логики первого порядка - это неудобно, и требует внимания.
2. Оптимизация представлений модели на языке SMT решателя. От оптимальности представления системы зависит быстрота поиска контр примера.

Список литературы

[1]. Pettersson J., Edström R. Safer smart contracts through type-driven development. Master's thesis, Chalmers University of Technology, Department of Computer Science and Engineering, Sweden, 2016.
[2]. Atzei N., Bartoletti M., Cimoli T. A survey of attacks on ethereum smart contracts (sok). Lecture Notes in Computer Science, vol. 10204, 2017, pp. 164-186.
[3]. Bhargavan K. et al. Formal verification of smart contracts. In Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91-96.
[4]. Luu L. et al. Making smart contracts smarter. In Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.
[5]. Hirai Y. Defining the ethereum virtual machine for interactive theorem provers. Lecture Notes in Computer Science, vol. 10323, 2017, pp. 520-535.
[6]. Kalra S. et al. Zeus: Analyzing safety of smart contracts. In Proc. of the Network and Distributed System Security Symposium, 2018.
[7]. Mueller B. Smashing Ethereum Smart Contracts for Fun and Real Profit. In Proc. of the 9th Annual HITB Security Conference, 2018.
[8]. Zakrzewski J. Towards verification of Ethereum smart contracts: a formalization of core of Solidity, Lecture Notes in Computer Science, vol. 11294, 2018, pp. 229-247.
[9]. Solidity github. Доступно по ссылке: https://github.com/ethereum/solidity/blob/develop/docs/grammar.txt, дата обращения 20.11.2018.
[10]. Nakamoto S. Bitcoin. A peer-to-peer electronic cash system. 2008. Доступно по ссылке: https://bitcoin.org/bitcoin.pdf, дата обращения 20.11.2018, дата обращения 20.11.2018..
[11]. Gideon Greenspan. Smart contracts and the dao implosion. 2016. Доступно по ссылке: https://www.multichain.com/blog/2016/06/smart-contracts-the-dao-implosion/, дата обращения 20.11.2018.
[12]. S. Palladino. The Parity Wallet Hack Explained, https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7, дата обращения 20.11.2018.
[13]. J.D. Alois. Ethereum Parity Hack May Impact ETH 500,000 or $146 Million. 2017. Доступно по ссылке: https://www.crowdfundinsider.com/2017/11/124200-ethereum-parity-hack-may-impact-eth-500000-146-million/, дата обращения 20.11.2018.
[14]. Jentzsch C. Decentralized autonomous organization to automate governance. 2016. Доступно по ссылке: https://download.slock.it/public/DAO/WhitePaper.pdf, дата обращения 20.11.2018.
[15]. Е. Шишкин. О построении среды для конструирования гарантированно надёжных смарт-контрактов. Материалы конференции РусКрипто'2018, 2018. Доступно по ссылке: https://www.ruscrypto.ru/resource/archive/rc2018/files/03_Shishkin.pdf, дата обращения 20.11.2018.
[16]. De Moura L., Bjørner N. Z3: An efficient SMT solver. Lecture Notes in Computer Science, vol. 4963, 2008, pp. 337-340.
[17]. Manna Z., Pnueli A. The temporal logic of reactive and concurrent systems: Specification. Springer-Verlag, 1992, 427 p/
[18]. Ethereum project. Доступно по ссылке: https://www.ethereum.org/, дата обращения 20.11.2018.
[19]. Szabo N. Smart contracts. 1994. Доступно по ссылке: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOT

winterschool2006/szabo.best.vwh.net/smart.contracts.html, дата обращения 20.11.2018.
[20]. Sheeran M., Singh S., Stålmarck G. Checking safety properties using induction and a SAT-solver. Lecture Notes in Computer Science, vol. 1954, 2000, pp. 127-144.

Verifying functional properties of smart contracts using symbolic model-checking

E.S. Shishkin <evgeniy.shishkin@gmail.com>
Infotecs, Scientific Research Department
1/23, Petrovsko-Razumovskiy Proezd, Moscow, 127287, Russia

Abstract. We describe our efforts towards building a tool that automatically verify high-level functional properties of Ethereum smart contracts against its formal specification that can be given using four different methods: an invariant over contract state or three different types of trace properties. A model of runtime system, the source code of smart contract together with its specification is translated into SMT-solver formula and checked for counter example. We tested the method on simplified version of notorious TheDAO smart-contract, called Mini-DAO. Our proof-of-concept tool was able to find a functional property violation of MiniDAO in just several seconds. We believe that the proposed method is indeed useful and deserves deeper investigation.

Keywords: symbolic model-checking; smart contracts; blockchain; formal specification;

DOI: 10.15514/ISPRAS-2018-30(5)-16

For citation: Shishkin E.S. Verifying functional properties of smart contracts using symbolic model checking. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 265-288 (in Russian). DOI: 10.15514/ISPRAS-2018-30(5)-16

References

[1]. Pettersson J., Edström R. Safer smart contracts through type-driven development. Master's thesis, Chalmers University of Technology, Department of Computer Science and Engineering, Sweden, 2016.
[2]. Atzei N., Bartoletti M., Cimoli T. A survey of attacks on ethereum smart contracts (sok). Lecture Notes in Computer Science, vol. 10204, 2017, pp. 164-186.
[3]. Bhargavan K. et al. Formal verification of smart contracts. In Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91-96.
[4]. Luu L. et al. Making smart contracts smarter. In Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.
[5]. Hirai Y. Defining the ethereum virtual machine for interactive theorem provers. Lecture Notes in Computer Science, vol. 10323, 2017, pp. 520-535.

[6]. Kalra S. et al. Zeus: Analyzing safety of smart contracts. In Proc. of the Network and Distributed System Security Symposium, 2018.
[7]. Mueller B. Smashing Ethereum Smart Contracts for Fun and Real Profit. In Proc. of the 9th Annual HITB Security Conference, 2018.
[8]. Zakrzewski J. Towards verification of Ethereum smart contracts: a formalization of core of Solidity, Lecture Notes in Computer Science, vol. 11294, 2018, pp. 229-247.
[9]. Solidity github. Available at: https://github.com/ethereum/solidity/blob/develop/docs/grammar.txt, accessed 20.11.2018.
[10]. Nakamoto S. Bitcoin. A peer-to-peer electronic cash system. 2008. Available at: https://bitcoin.org/bitcoin.pdf, accessed 20.11.2018, accessed 20.11.2018..
[11]. Gideon Greenspan. Smart contracts and the dao implosion. 2016. Available at: https://www.multichain.com/blog/2016/06/smart-contracts-the-dao-implosion/, accessed 20.11.2018.
[12]. S. Palladino. The Parity Wallet Hack Explained, https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7, accessed 20.11.2018.
[13]. J.D. Alois. Ethereum Parity Hack May Impact ETH 500,000 or $146 Million. 2017. Available at: https://www.crowdfundinsider.com/2017/11/124200-ethereum-parity-hack-may-impact-eth-500000-146-million/, accessed 20.11.2018.
[14]. Jentzsch C. Decentralized autonomous organization to automate governance. 2016. Available at: https://download.slock.it/public/DAO/WhitePaper.pdf, accessed 20.11.2018.
[15]. Shishkin E. Towards building an environment for reliable smart contracts construction. RusCrypto, 2018.
[16]. De Moura L., Bjørner N. Z3: An efficient SMT solver. Lecture Notes in Computer Science, vol. 4963, 2008, pp. 337-340.
[17]. Manna Z., Pnueli A. The temporal logic of reactive and concurrent systems: Specification. Springer-Verlag, 1992, 427 p/
[18]. Ethereum project. Available at: https://www.ethereum.org/, accessed 20.11.2018.
[19]. Szabo N. Smart contracts. 1994. Available at: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, accessed 20.11.2018.
[20]. Sheeran M., Singh S., Stålmarck G. Checking safety properties using induction and a SAT-solver. Lecture Notes in Computer Science, vol. 1954, 2000, pp. 127-144.