Automation of microservices creation for qualitative analysis of binary dynamic systems

G A Oparin, V G Bogdanova and A A Pashinin

Matrosov Institute for System Dynamics and Control Theory SB RAS, Lermontov St. 134, Irkutsk, Russia, 664033

bvg@icc.ru

Abstract. The main objective of qualitative research is to analyze the behavior of the trajectories of a dynamic system to verify whether it corresponds to the set of constraints characterizing the property. We use an approach to study binary dynamic systems on a finite time interval based on the author's method of Boolean constraints. Based on this method, the Boolean model of the properties of a binary dynamic system is written in the language of Boolean equations or Boolean formulas with quantifiers. Thus, the verification of various dynamical properties is reduced to solving the problems of Boolean constraints satisfiability or the validity of a quantified Boolean formula using efficient SAT or TQBF solvers. The high computational complexity of these problems requires the development of software and tools for their parallel and distributed solving and ensuring transparent end-user access to high-performance computing environments based on a service-oriented approach. This paper represents the architecture and functionality of a new instrumental system that automates the creation of a distributed application for solving the considered class of problems based on the microservice approach and multi-agent technology.

1. Introduction

The extensive use of binary dynamic systems (BDS) in both scientific and applied researches determines the relevance of developing new and improving existing methods for qualitative analysis of the BDS trajectories behavior. The high computational complexity of these problems requires the development of software and tools to solve it using technologies of parallel and distributed computing and ensuring clear end-user access to resources of high-performance computing environments based on a service-oriented approach.

An approach based on the method of Boolean constraints [1] to study BDS is used for solving some qualitative analysis problems represented in [2, 3]. Based on this method, the Boolean model of the dynamical property of BDS satisfying logical property specification and equations of system dynamics is written in the language of Boolean equations or Boolean formulas with quantifiers. The satisfiability verification of a property is reduced to solving the problems of satisfiability of Boolean constraints or verification of the validity of a quantified Boolean formula using efficient SAT or TQBF solvers.

Existing software for qualitative analysis of autonomous synchronous BDS (also called Boolean network) [4-7] is mainly intended only for searching attractors. It is noted in [8] that these tools have several drawbacks, in particular, they are limited by the complexity of the Boolean model and the format of its presentation, require programming skills from subject specialists, since they are often used only as command line tools depending on the platform. Thus, it is relevant to develop new methods that provide the ability to solve problems of qualitative analysis of BDS for large dimensions of the state vector on a sufficiently long time interval and parallel software tools for implementing this method. The advantages of the author's Boolean constraints method in comparison with the existing ones are given in [1].

Therefore, during the solving of qualitative analysis problems based on the method of Boolean constraints, the following software is required:

- Tools for constructing a Boolean model both for searching for attractors and for specifying various dynamic properties of BDS (in particular, reachability, isolation, attraction, connectivity) in the required format;
- Efficient parallel solvers for SAT and TQBF problems.

In the context of the proposed approach, software tools for constructing a Boolean model that specifies a dynamic property and solutions of the resulting system of Boolean equations are implemented as independently applied microservices. The decentralized management of the interaction of microservices is carried out by a self-organizing multiagent system, agents of which are delegated the rights to launch microservices. To automate the creation, deployment, and testing of microservices, the previously developed HPCSOMAS toolkits [9, 10] is extended by a new specialized subsystem MSCDT (Micro-Services Creating, Deployment and Testing), the architecture and functionality of which are considered in this work. The focus is on the deployment of microservices.

2. Related work

Currently, the direction associated with the development of distributed applications based on microservice architecture is actively developing. The advantages and disadvantages of microservice-oriented applications in comparison with monolithic ones, issues related to the design, creation, and deployment of microservice-oriented software systems are discussed in detail in [11, 12]. Nowadays, the trend of creating automation tools for processes of microservices deploying and testing is observed. In [13], the analysis of existing testing methods, that are used for such distributed systems, is performed, and a new model of the microservice system validation is represented. In [14], an approach for the automatic generation of self-configuring microservice regarding the target execution environment was proposed. In [15], still open problems are given, in particular, those related to the automation of microservices configurations choice and their mapping to heterogeneous computing environment and the specification of the topology and composition of the microservice application. Consequently, a relevant research area is a study related to the development of tools that ensure the solving of such problems, namely, the formation of a composition of microservices, their reuse and simplified interaction with a complex computing infrastructure.

The dynamic nature of cloud computing environment and the complexity arising in the research in the subject area of the exhaustive problems of qualitative analysis leads to the intensification of the development of microservice-oriented software tools for their solution based on self-organization and multi-agent approach [16, 17]. An application of multi-agent technology in the implementation of microservice architecture is described in [18, 19].

Based on direct interactions of agents management by a microservices ensemble provides better adaptability to dynamic environments and higher reactivity to external influences compared to indirect [20, 21]. Taking into consideration the disadvantages and advantages of the above approaches, we use a self-organizing multi-agent system (MAS) to organize decentralized management.

Instead the description of microservices choreography in programming language, in contrast to existing works, our approach uses the discrete event model of the functioning of MAS agents, which delegates the right to launch microservices. This model was developed by authors in [10]. The MSCDT subsystem provides automation of the agents configuration.

3. MSCDT architecture

Reactive agents based on software modules (Computational Module Agent, CMA) and intelligent distributed solver agents (DSA) are created in the form of microservices with the help of MSCDT. The MSCDT subsystem automates the configuration of CMA and DSA agents. MSCDT also provides tools for automating the deployment and testing of computational microservices and filling the local knowledge bases (KB) of DSA agents that are delegated the rights to launch these microservices on dedicated computing resources of a heterogeneous distributed computing environment (figure 1). A computational domain model is used as the KB, which is distributed the way that each agent has limited knowledge of both the capabilities of other system agents and the computational field (CF) topology as a whole. The local KB stores the data about relationships with neighboring agents.

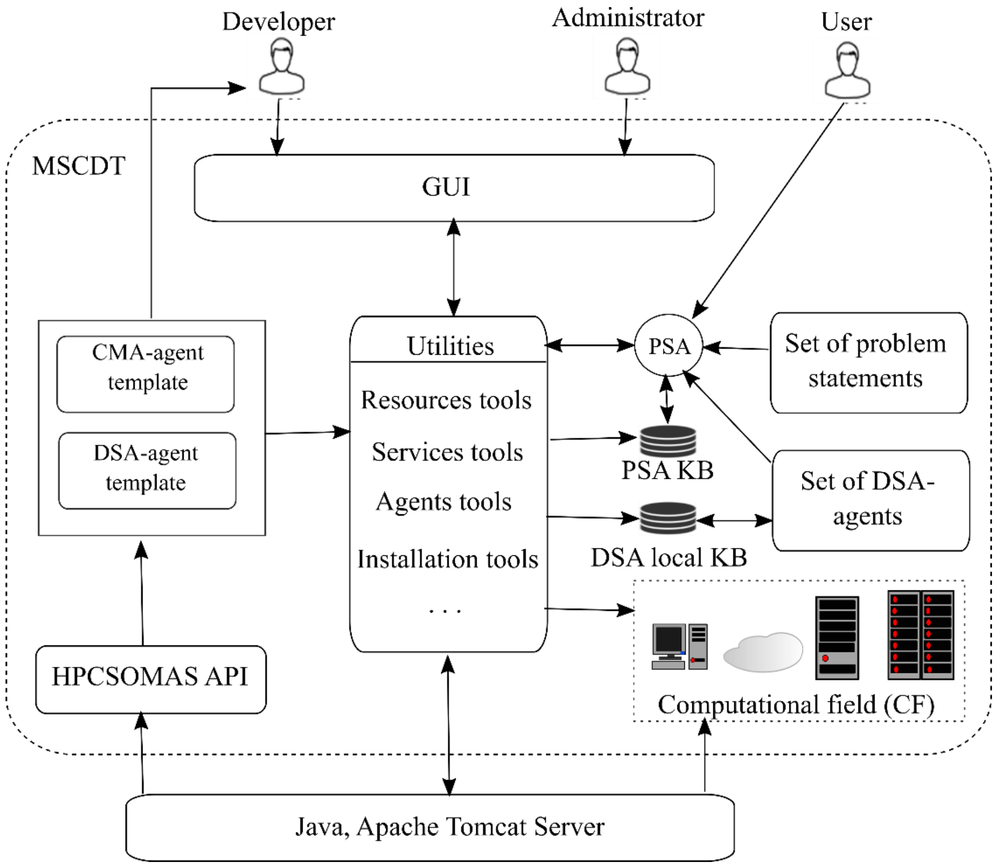


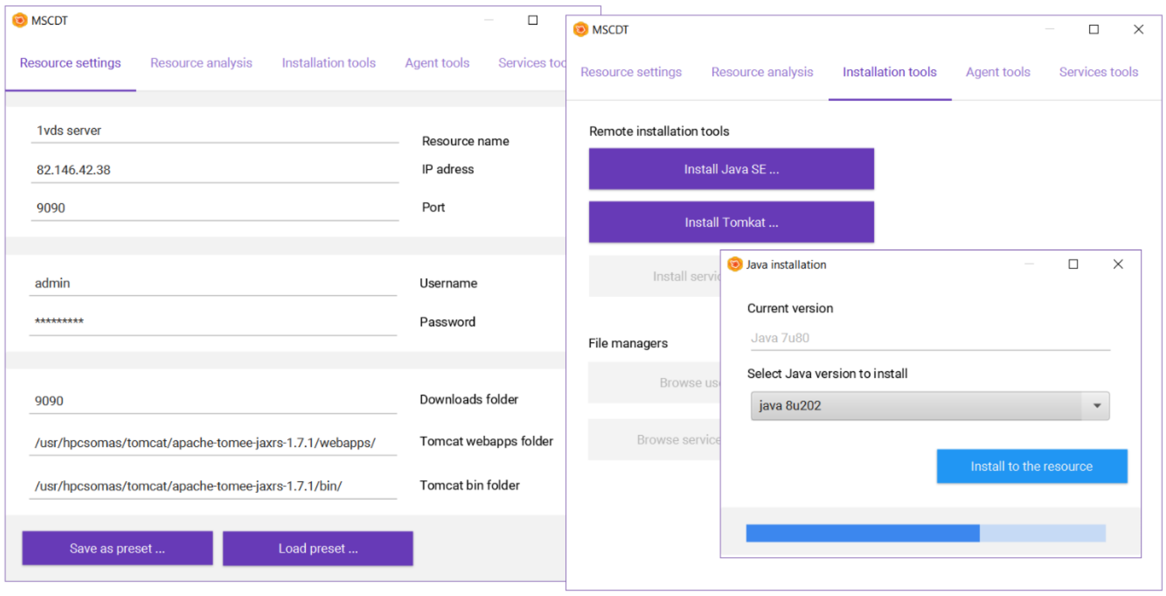
Figure 1. MSCDT architecture.

The computational model is represented as a set of domain parameters and functional relationships between them. Each functional relation is implemented by a software module that calculates the values of the output parameters according to the specified values of the input parameters. A CF is a set of network-connected logical computing nodes on which agents of a distributed solver are installed. The functionality of each agent is determined by the requirements of inclusion of the module associated with the agent in the computational process of solving an applied problem. A logical node is a physical computing resource, which can be: a set of processor cores and nodes of a computing cluster, a personal computer, a virtual machine, a mobile device. CF is discrete. In each node, the CF numeric value is calculated, taking into account the node's agent state and according to the rule of CF agents local interactions. The initial CF value of the node of DSA is set equal to the number of signs of computability transmitted by the predecessor agents and decreases as they are received. The zero value of the CF is a trigger for the inclusion of the corresponding module in the process of solving an applied problem.

For the non-procedural formulation of the “Given-Find” problem [3] on the distributed computational domain model, the Problem Statement Agent (PSA) web-interface is provided. The KB of PSA agent stores the relations of the CF modules and nodes. In distributed and cloud computing, the PSA agent is installed on dedicated computing nodes and is the entry point to the system.

3.1. *Microservices development*

During the microservice creating, the developer has to perform many different routine operations, especially in experimental testing. The need for this operation arises after updating the developed software that implements the functions of the microservice. For example, a compiled microservice needs to be moved to a correspondent directory of the Tomcat services server by connecting to a computational resource via the SSH protocol. In some cases, it is necessary to suspend the server using console commands on this resource. The console commands, location and working directories names for the installed software depend on this resource. During the microservices creating, the developer also needs to configure agents and environment variables. For example, on different computing resources, there are different paths to the directories for downloading files, storing executable files, user directories, and the same microservice will need to be configured differently. Similar problems arise during the resources administering, adding new, and updating existing computational microservice and DSA agents. Access to the functionality of these utilities is reached through a graphical user interface (GUI) shown in figure 2.



1vds server	Resource name
82.146.42.38	IP adress
9090	Port
admin	Username
*********	Password
9090	Downloads folder
/usr/hpcsomas/tomcat/apache-tomee-jaxrs-1.7.1/webapps/	Tomcat webapps folder
/usr/hpcsomas/tomcat/apache-tomee-jaxrs-1.7.1/bin/	Tomcat bin folder

Current version: Java 7u80

Select Java version to install: java 8u202

Figure 2. GUI of MSCDT system.

The “Service tools” menu is intended for creating microservices. The developer of computational microservices can use two approaches for this: programming using Java language and the class library HPCSOMAS-MS API (Applied Program Interface); usage of standard compiled implementations (templates) of microservices. In the second case, the developer only needs to change the parameters in the configuration file. In this case, the developer is freed from the study of the low-level implementation details of creating and functioning microservices.

Both approaches assume that the developer will need to load the services he creates onto resources for testing, debugging, and providing to users. Subsystem MSCDT allows simplifying these processes. Having data for authorization and access granted, the developer can upload microservices and related software to a resource where PSA agents are already operating, configure it to work in the environment of this resource and prepare agents for working with installed microservices. The “agent tools” menu is used to configure agents. PSA-agents are pre-installed on a computational resource using the same tool by the administrator. The software necessary for the functioning of these agents, in particular, Java SE and Tomcat, is installed using the “Installation tools” menu. The “Resource analysis” and “Resource setting” menus are used to set up a computational resource.

The developer or administrator once fills in the parameters of the resource and saves them as presets to quickly switch between them. The utility package performs the following actions in automatic mode:

- Uploading the compiled service to the server directory of the services on the computing resource;
- Possible suspension of the service server at the time of service replacement to eliminate unwanted automatic deployment before additional actions with files;
- Editing service configuration files;
- Loading configuration files associated with this computing resource;
- Automatic reconfiguration of these files of existing microservice to work with a given computing resource (replacement of environment directories, resource parameters);
- Adding a microservice to the PSA agent registry of this resource;
- Remote tracking of the service workability, analysis of error reports;
- Transmission of microservices description to dependent microservices.

MSCDT allows deploying the microservices on a wide range of computing resources, including computing clusters with job control systems, virtual dedicated servers in the cloud (VDS), containers in virtualization systems (Docker) and individual workstations (figure 3).

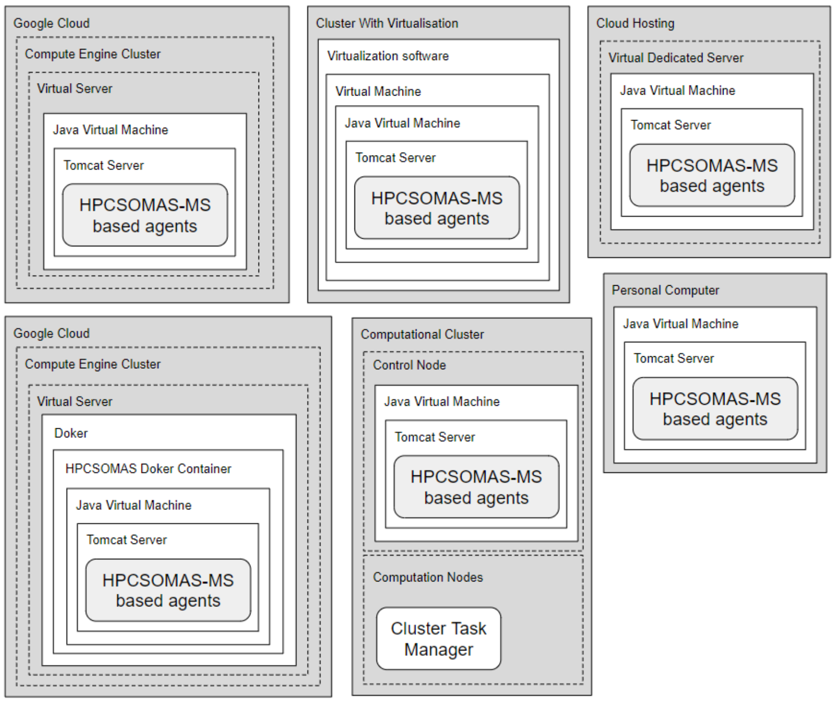


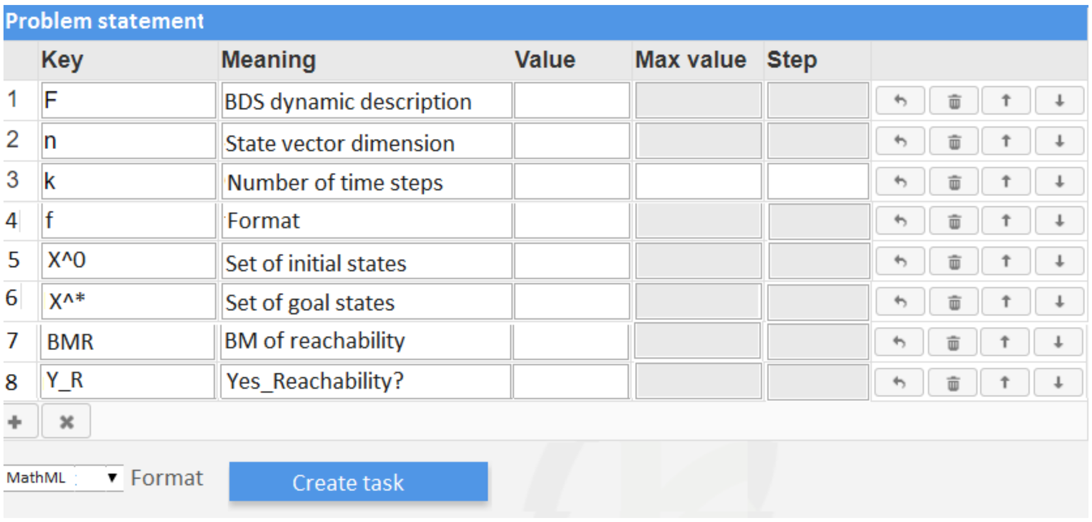
Figure 3. Variants for deploying microservices.

Since the implementation of microservices is based on Java Servlet technology, they can work on any resources on which Apache Tomcat Server can be installed. The choice of a resource depends on the required computational characteristics and its availability as well as on the system requirements of the application programs installed on the DSA agent launching the computational microservice. The resource type also determines the degree of the deployment process automation.

In some cases, DSA agents can be automatically installed and configured using the SSH protocol. In other cases, in particular, in the Google Compute Engine, DSA agents are deployed using a correspondent set of platform tools for computing resources, namely the Google Cloud SDK for this case.

3.2. PSA agent interface.

The end user is provided with a PSA-interface for forming a request for solving the problem. For the non-procedural formulation of the problem, the “Problem Statement” menu is used. The user selects the subject area. As a result, a list of parameters appears, where the required ones in the “Value(In)” and “Value(Out)” columns should be ticked (figure 4). Then the task is created. Task execution stages are considered in [3] in detail. First, an active group of DSA agents is formed for executing this task by the logical inference using the non-procedural formulation of the problem over the distributed model of the subject area. After this stage in the process of solving the problem, two situations may arise. In the first case, an active group of agents will be formed (which will ensure the calculation of the output data values by the given input data values). In this case, the grey colored fields (figure 4) will be accessed for entering the parameter values. Then the task will be solved at the stage of joint actions of the DSA agents. In the second case, the user is informed that this task is unsolved. PSA agent also is intended to process the following objects: the dictionary of domain parameters, microservices that implement the functionality of applied subject modules, logical CF nodes associated with a specific computational resource, and the list of agents delegated to launch the modules. Only HPCSOMAS system administrator and microservice developer have access to these PSA agent objects.



Problem statement

	Key	Meaning	Value	Max value	Step
1	F	BDS dynamic description			
2	n	State vector dimension			
3	k	Number of time steps			
4	f	Format			
5	X^0	Set of initial states			
6	X^*	Set of goal states			
7	BMR	BM of reachability			
8	Y_R	Yes_Reachability?			

MathML Format

Create task

Figure 4. Problem statement interface.

4. MSCDT-based on Boolean modelling system for qualitative analysis of BDS

A subject-oriented subsystem MSQABDS (Micro-Services based Qualitative Analysis of Binary Dynamic System) (figure 5) is developed using MSCDT and intended to automate building Boolean models of the BDS dynamical properties.

Models of dynamic properties in the form of a Boolean constraint, satisfying the logical specification of the property and the equations of the dynamics of the BDS are constructed for autonomous synchronous BDS, the vector – matrix equation of which has the form

$$x^t = F(x^{t-1}), \tag{1}$$

where x is the state vector, $x \in B^n$, $B = \{0,1\}$, n is the state vector dimension; $t \in T = \{1,2,\ldots,k\}$ is discrete time (the time step number); $F(x)$ is a vector function of Boolean algebra which is call transition function.

The MSQABDS system includes the microservices for constructing the Boolean constraint based on the system dynamics equations (1), these equations may be specified in CNET [5] or MathML [22] formats. The basis for constructing a Boolean model of a dynamic property is the Boolean one-step transition formula obtained according to (1) (the left side of the equation $x^1 \oplus F(x^0) = 0$).

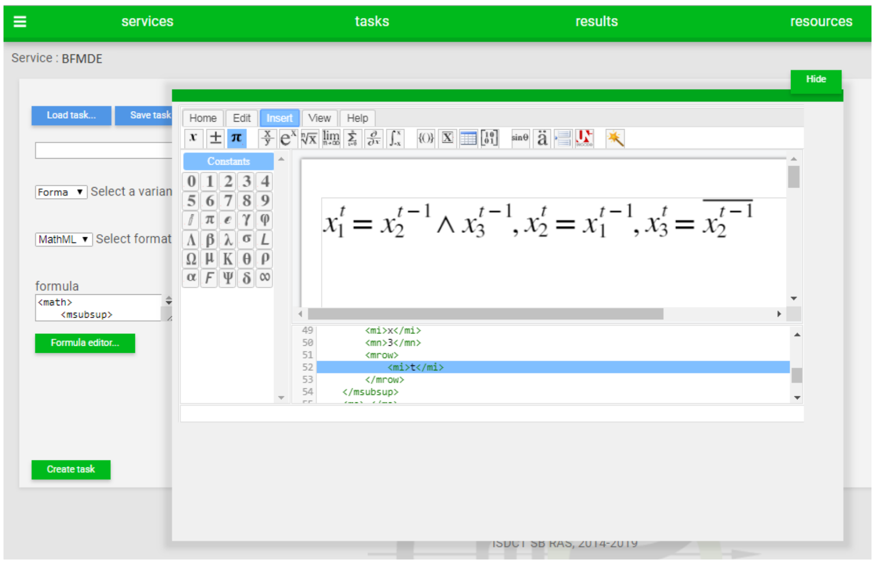
The Boolean editor uses this one-step transition formula and a property specification, which may include the initial, admissible, and target state sets for building a Boolean property model. The converters, the editor and pre-processor are implemented as microservices based on ready-made templates (based on the class for the CMA agent from the HPCSOMAS API class library) of the MSCDT subsystem. Using the MSCDT GUI, the user can select the preferred template and edit it using the "Agents tools" menu (figure 5).

4.1. Microservice of editing

The subsystem of the Boolean modelling includes microservices that performs the following functions:

- Loading a dynamics description of the system (1) (DD) in a valid input format;
- The typing of the DD in a mathematical language using the freely distributed FMath Javascript Equation Editor [23];
- Converting the DD into an output format using suitable conversion microservices;
- Saving the DD in the mathematical language using the formats Latex or MathML.

A screenshot of the microservice of editing is shown in figure 6.



$$x_1^t = x_2^{t-1} \wedge x_3^{t-1}, x_2^t = x_1^{t-1}, x_3^t = \overline{x_2^{t-1}}$$

```
<math>
    <msubsup>
```

```
        <mi>x</mi>
        <mn>3</mn>
        <mrow>
            <mi>t</mi>
        </mrow>
    </msubsup>
```

Figure 6. Microservice of editing.

4.2. Microservice of converting MathML-DIMACS

The MathML-DIMACS converter is used during building the Boolean model of the dynamical property of BDS and has the following features:

- Conversion of a Boolean function from MathML format to an internal representation;
- Analysis of the Boolean function structure, revealing internal parallelism;
- Construction of a parallel plan for calculating the Boolean function;

- The dynamical constructing of the truth table with simultaneous generating the Boolean function representation in using the DIMACS format.

The output language (MathML) of the formula editor is used as the input language of the converter. The converter reads the Boolean function, performs lexical and syntactic analysis, and forms the internal representation of the Boolean function. The Boolean function is represented as a labeled tree. The leaves are marked with symbols from an alphabet containing variable names. Other vertices are marked with symbols from the alphabet of symbols of Boolean operations. The vertex of the tree is represented in the computer memory by a recursive data structure containing a field for storing the computed value (of this subtree), type (variable or operation) and references to the parent and children.

The planner bypasses the tree and builds a layer-parallel form of the algorithm for calculating this function. This algorithm calculates the length of the maximum path from the vertex to the each leave of the tree. Then vertices are distributed between layers so that these paths must be the equal length at one tier. Thus, the height of the layer-parallel form cannot be larger than the height of the tree. Then the vertices of each layer are divided into independent blocks that can be processed in parallel.

A parallel plan of calculation for the calculator is formed. The parallel calculator of the Boolean function is applicable for dynamical building a truth table, using which the Boolean constraint in DIMACS format is generated. MSQABDS is an open system that allows expanding the set of input formats with the inclusion of an additional converter.

4.3. Experimental study

When solving problems of a qualitative study of BDS based on the method of Boolean constraints, we use a declarative approach, in contrast to the procedural one presented, for example in [4, 5]. Based on the declarative approach, the satisfiability of the required dynamical property of the system (1) is represented as some set of Boolean constraints (or Boolean model) on the behavior of the BDS trajectories. The integration in one model of both the description of the property and the equations of the dynamics (1) of a specific object is the distinctive feature of such models developed using the method of Boolean constraints.

The proposed declarative approach implies a uniform automated scheme for constructing models for different dynamic properties in the form of Boolean constraints. Satisfiability of these constraints is verified using an effective Sat solver. Thus, the same solver may be used to check different properties. For example, based on offered approach Boolean models for following dynamical properties of the reachability type is represented in [1]: the main reachability property; the security property; the simultaneous reachability property; the reachability property under phase constraints; the attraction property; the connectedness property of the target set; the total reachability property of the target set from the set of initial states.

The procedural approach involves the implementation of an algorithm for verifying the satisfiability of the required dynamic property in a programming language. This approach, unlike the declarative one, firstly, requires another algorithm to check other property. Secondly, such algorithms are not parallelized well [8]. Thirdly, most of the algorithms are designed only to search for attractors. In this case, additional complexity arises - the search for all solutions is required.

In the first computational experiment, a comparison of based on these approaches computations were carried out for the problem of searching for equilibrium states in autonomous synchronous Boolean networks (models of gene regulatory networks – the GRN, most often used in testing [4, 7]) shown in table 1.

Based on MSQABDS system, for these GRNs, Boolean models that describe the equilibrium state for a specific description of the dynamics of the BDS were constructed. The equilibrium states are satisfiable sets of Boolean constraints of the model, which are found with the help of the “AllSAT solution” solver [24]. In the declarative approach, the following “AllSAT solution” solvers are used to search for the equilibrium states of the constructed Boolean models: nbc_minisat_all-1.0.2 and bc_minisat_all-1.1.2 [24]. In the procedural approach, the BNS solver (bns_v1.3) [4] was used. BNS is an SAT-based program (not the SAT solver) for searching for cycles of states (attractors) in Boolean

Networks with Synchronous update (a free version provided on https://people.kth.se/~dubrova/bns.html). This well-known solver is still used for comparison with new analogous software tools developed in recent years [6, 7]. The input format of BDS dynamics description (1) for this solver is CNET format. The speedup obtained when solving the problem for equilibrium states searching using the declarative approach, in comparison with the procedural one, is shown in figure 7.

Table 1. An example of GRNs.

GRN	Nodes	Equivalent states	Dynamic description in “cnet” format
Fission yeast	10	13	https://people.kth.se/~dubrova/BNS/fission_yeast.cnet
Mammalian cell	10	1	https://people.kth.se/~dubrova/BNS/mammalian.cnet
Arabidopsis Thaliana	15	10	https://people.kth.se/~dubrova/BNS/arabidopsis.cnet
T-helper cell	23	3	https://people.kth.se/~dubrova/BNS/thelper.cnet
T-cell receptor	40	8	https://people.kth.se/~dubrova/BNS/tcr.cnet
Drosophila melanogaster	52	7	https://people.kth.se/~dubrova/BNS/drosophila4.cnet

In the second computational experiment, the search for equilibrium states was carried out for the BDS of the form

$$x_1^t = x_n^{t-1}, x_2^t = x_1^{t-1}, \dots, x_i^t = x_{i-1}^{t-1}, \dots, x_n^t = x_{n-1}^{t-1}, \tag{2}$$

where n is the dimension of the BDS state vector.

Boolean models for the problem of searching for equilibrium states were constructed using MSQABDS in DIMACS format (for nbc_minisat_all- and bc_minisat_all-solver) and generated in CNET format (for BNS solver) for various values of n, increasing from 100 to 15,000. The results of solving the problem under consideration are shown in figure 8. The speedup resulting from the declarative approach steadily increases for both nbc_minisat_all- and bc_minisat_all-solver in comparison with BNS solver when increasing dimension n of the state vector.

All experiments are performed on the HPC-cluster “Akademik V.M. Matrosov” (nodes with two 16-core processors AMD Opteron 6276 «Bulldozer»/«Interlagos» 2.3 GHz, 16 MB L3 cache, 4 FLOP/cycle).

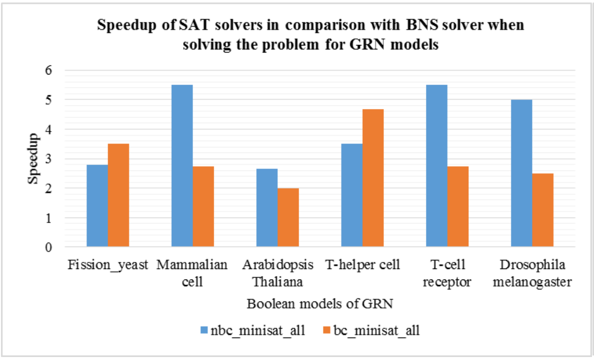


Figure 7. Comparison of solvers for search of equivalent states in GRN (table 1).

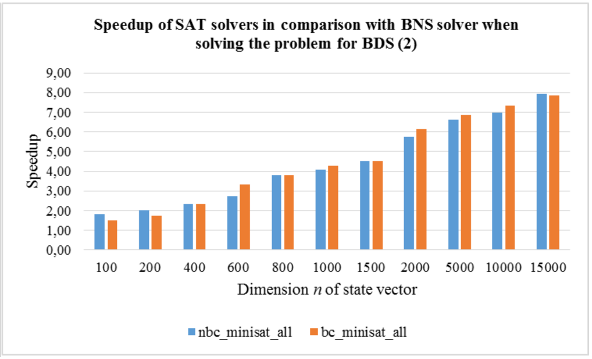


Figure 8. Comparison of solvers for search of equivalent states in BDS (2).

Reducing the solution of qualitative analysis problems to solving SAT and 2QBF problems allows data parallelism and provides the possibility for study the trajectories behavior for BDS of large dimensions n of the state vector over a sufficiently long time interval T.

5. Conclusion

We offer an approach to automate the development of microservice-oriented applications using multi-agent technology. Based on this approach, we develop the MSCDT for automating the creation, deploying, and testing microservices. The microservice-oriented system MSQABDSA for building the Boolean model of a BDS dynamical property is created using the MSCDT. MSQABDSA system is intended for solving the problems of qualitative analysis of BDS. The widespread use of BDS as models of gene regulatory networks determines the practical relevance of the tools developed.

Acknowledgment

This work was supported by the Russian Foundation of Basic Research, project no. 18-07-00596 (reg. no. № AAAA-A18-118012290008-8). This work was also supported in part by the Presidium RAS, program no. 2, project “Methods and tools for solving hard-search problems with supercomputers” (reg. no. AAAA-A18-118031590005-5). The authors would like to thank the Irkutsk Supercomputer Center of SB RAS for providing access to cluster computational resources.

References

[1] Oparin G A, Bogdanova V G and Pashinin A A 2018 Boolean constraints method in qualitative analysis of binary dynamic systems *International Journal of Applied and Basic Research* **9** 19-29 (In Russian)

[2] Bychkov I V, Oparin G A, Bogdanova V G and Pashinin A A The applied problems solving technology based on distributed computational subject domain model: a decentralized approach *Parallel computational technologies* (Chelyabinsk: SUSU) pp 34-48

[3] Oparin G A, Bogdanova V G, Pashinin A A and Gorsky S A Distributed solvers of applied problems based on microservices and agent networks 2018 *Proc. of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics* (IEEE) pp 1415-1420

[4] Dubrova E and Teslenko M 2011 A SAT-based algorithm for finding attractors in synchronous Boolean networks *Trans. Comput. Biol. Bioinformatics* (IEEE/ACM) **8**(5) 1393-1399

[5] Dubrova E, Teslenko M and Martinelli A 2005 Kauffman networks: analysis and applications *Proc. of the Int. Conf. on Computer-Aided Design* pp 479-484

[6] He Z, Zhan M, Liu S, Fang Z and Yao C 2016 An algorithm for finding the singleton attractors and pre-images in strong-inhibition Boolean networks *PLOS ONE* **11**(11) e0166906

[7] Zheng D, Yang G, Li X, Wang Z and Hung W N 2013 An efficient algorithm for finding attractors in synchronous Boolean networks with biochemical applications *Genetics and Molecular Research* **12**(4) 4656-66

[8] Guo W, Yang G, Wu W, He L and Sun M 2014 A parallel attractor finding algorithm based on Boolean satisfiability for genetic regulatory networks *PLOS ONE* **9**(4) e94258

[9] Bychkov I V, Oparin G A , Bogdanova V G , Pashinin A A and Gorsky S A 2017 Automation development framework of scalable scientific web applications based on subject domain knowledge *Parallel Computing Technologies* ed V Malyshkin (Springer: Cham) LNCS **10421** pp 278-288

[10] Bychkov I V, Oparin G A, Bogdanova V G and Pashinin A A 2018 Service-oriented technology for development and application of decentralized multiagent solvers for applied problems *Herald of computer and information technologies* **12** 36-44 (In Russian)

[11] Richardson C and Smith F 2016 Microservices - from design to deployment (San Francisco: NGINX)

[12] Newman S 2015 Building Microservices (O’Reilly)

[13] Savchenko D and Radchenko G. Microservices 2015 Validation: Methodology and implementation *Proc. of the`1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists* 1513

[14] Kehrer S and Blochinger W 2018 AUTOGENIC: Automated generation of self-configuring

microservices *Proc. of the 8th International Conference on Cloud Computing and Services Science* (SCITEPRESS) pp 35-46

[15] Fazio M, Celesti A, Ranjan R, Liu C, Chen L and Villari M 2016 Open issues in scheduling microservices in the Cloud *Cloud Computing* (IEEE) **3**(5) pp 81-88

[16] Gorodetskii V I 2012 Self-organization and multiagent systems: I. Models of multiagent self-organization *Journal of Computer and Systems Sciences International* **51**(2) 256–281

[17] Rodrigues N 2014 Dynamic composition of service oriented multi-agent system in self-organized environments *Proc. of the Workshop on Intelligent Agents and Technologies for Socially Interconnected Systems* pp 1-6

[18] Oberhauser R 2016 Microflows: lightweight automated planning and enactment of workflows comprising semantically-annotated microservices *Proc. of the Sixth Int. Symposium on Business Modeling and Software Design* pp 134-143

[19] Florio L 2015 Decentralized self-adaptation in largescale distributed systems *Proc. 10th Joint Meeting on Foundations of Software Engineering* (ACM) pp 1022-1025

[20] Buguillo J 2018 *Self-organizing Coalitions for Managing Complexity* (Springer International Publishing) pp 89-100

[21] Moussaid M 2009 Collective information processing and pattern formation in swarms, flocks, and crowds *Top Cogn Sci* ed M Moussaid et al. **1**(3) pp 469–497

[22] Kohlhase M and Rabe F 2012 *Math.Comput.Sci* **6**(3) (Springer Basel AG) 235–260

[23] Alexandru I 2017 FMath editor (*Electronic Materials* https://www.fmath.info/, accessed: May, 20 2019)

[24] Toda T and Soh T *ACM Journal of Experimental Algorithmics* 2016 **21**(1)