

Intellectual technology for computation control in the package of applied microservices

I V Bychkov, 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 complexity of exhaustive problems with the properties of large-scale, openness, unpredictable dynamics, and component mobility determines the relevance of developing microservice-oriented software for their solving in a hybrid computational environment. We propose an approach for adapting to this environment both the existing software and new ones developed using new automated technology for creating an applied microservice package and organizing control of computations in it. The distributed computational model is represented by a set of small, loosely coupled, replaceable, interacting with the use of lightweight communication mechanisms autonomous microservices that implement the functions of the program package modules. The decentralized management of the microservices interaction is carried out by a self-organizing multi-agent system, agents of which are delegated the rights to launch microservices. The paper discusses the models, methods, and software platform that form the basis of the proposed technology. We demonstrate the application of the applied microservice package, based on this technology, for solving the problems of qualitative analysis of binary dynamic system using the author's Boolean constraints method.

1. Introduction

Recently, the trend of active use of the microservice technology for accessing cloud resources is observed mainly in the development of business applications. Currently, many papers (for example, [1, 2]) raise issues related to the development of tool environments and the providing infrastructures for implementation resource-intensive scientific applications based on microservices. Our research focuses on the use of microservice technology for automating both the creation new and adaption an existing application software for the hybrid computing infrastructure that provides the ability to solve complex problems of some subject domain (SD). The research in such SD requires computational experiments (multivariate calculations), due to the variation of the mathematical model of the problem, the use of various research methods, algorithms of implementation based on these methods, and source data. In particular, the area of our scientific interests is related to the study of dynamics and structural-parametric synthesis for different classes of dynamic controlled systems. The experience of development of the applied program packages in this area is summarized in [3]. Currently, we focus on the qualitative study of binary dynamic systems (BDS) based on the developed by authors method of Boolean constraints [4].

Based on the proposed technology, hybrid computing environment oriented software is presented in the form of a package of applied microservices (AMP). The AMP distributed computational model is represented by a set of small, loosely coupled, replaceable, and interacting with the use of lightweight communication mechanisms autonomous microservices [5], within our approach, implementing the functions of the package modules. Nowadays, the researchers pay great attention to

microservices-oriented computational infrastructure development. Thus, various aspects of a complete transition to the cloud structure, related to reliability and availability, the confidentiality of data stored in the cloud, and expenses are discussed and compared in [6]. In [2] the advantage of using on-premises computers and connecting to the cloud for scaling the computations in case of resources lack are noted. The objective of our research is to develop an automated technology for creating and supporting the AMP functioning in a hybrid computing environment that includes on-premises computers and, on demand, the resources of a public cloud. The new software platform underlying this technology is the further development of the author's tools [7]. Based on this platform new technology provides the automation for both the creation of AMP and the organization of decentralized control in the process of solving an applied problem based on direct interactions of agents (which delegates the right to perform microservices), providing better adaptability to dynamic environments and a higher reactivity to external influences compared to indirect ones. A modified knowledge base and the new architecture of control agent provide of the organizing the parallel pipeline in dynamic multivariate computations along with static multivariate one.

2. Related work

The necessity of development of scalable microservice-based applications instead of producing monolithic applications is discussed in [8, 9]. Unpredictable dynamics, mobility of components of some problems with complex structure, and variability of cloud computing environments for solving these problems actualize the usage of self-organization mechanisms and the multiagent approach [10, 11] for the development of software based on microservice-oriented architecture [12]. This paper argues that an agent can be viewed as a type of microservice that can be deployed seamlessly within any microservice ecosystem. The control organization in such systems requires new models, architectures, and development technologies [11]. In [13], it is noted that modeling both individual microservice components and their ensembles is challenging. In [14], the foundations of ensemble modeling are considered, in particular, the model for the organization of goal-oriented behavior of this ensemble is noted as a difficult problem. The engineering of these ensembles for providing their reliability in conditions of changes in ensemble environment and requirements is one of the most tasks [15]. The application of the multiagent system to decentralized self-adaptation of microservices based on Docker containers [16] is described in [17]. Distributed organization and decentralized multiagent management are indicated in [18-20] as among the main criteria for the quality and reliability of such software systems. Direct interactions of agents in comparison with indirect ones provide better adaptability to the variability of cloud environments and higher reactivity [21]. Microservices are considered as modern agents that could improve systems of interrelated computing devices in distributed environments, such as the Internet of Things in [22]. Unlike this, we focus on scientific computations.

The distinctive features of our approach are the following:

- The combination of microservices for providing research in specific SD is based on the AMP development principles;
- Goal-oriented behavior of the microservices ensemble for solving a problem in this SD is based on the non-procedural statement (NPS), decentralized control, self-organization, and knowledge base (KB);
- This ensemble (and correspondently an active group of agents launched these microservices) is formed by logical inference using NPS over distributed KB;
- The means of achieving the goal are a cooperative approach to solving the problem, direct semantic interactions of agents, and a discrete-event finite-state behavioral model of the agent.

3. Basic principles of AMP development and application

The methodological aspects of the proposed specialized technology for creating and using the AMP are the following: modularity and knowledge representation about mathematical models of studied SD

objects, methods, and methodology for their analysis and design; formulation of research problems; problem solving management.

Modularity is one of the central structural properties of complex models, methods, and methodologies of their study. For example, this principle is underlined in the mathematical model constructing in the research areas of dynamic analysis and structural-parametric synthesis of control systems. The method of solving a problem is focused, as a rule, on representing the model in some standardized form. In this case, algorithms are provided both for converting a model from one format to another and for constructing a mathematical model of a complex system without taking into account its structure by usage the describing of the models of its elements and connections between them. The modularity principle provides the replacement of programming way for method implementation with the design from ready-made, previously developed, autonomously translated and debugged modules, and allows the automatic ensemble of modules to solve a research problem.

3.1. Conceptual model of subject domain

Algorithmic knowledge of mathematical models, methods, and techniques for their study has a complex hierarchical modular structure represented by three conceptually distinct layers: productional, schematic, and computational. Concepts of the first layer are specified through (have references to) schematic layer concepts that are specified through the computational one. Computational knowledge is represented by a library of specified autonomous sub-programs. The schematic layer is a system of interconnected objects, namely operations (O), and parameters (P), which are the simplest, most adequate, and expressive means for describing the modular structure of a mathematical model and its algorithms for its analysis. The organization of intermodular interfaces requires supporting the symbolic name mechanism in the SD specification language and providing the uniqueness of parameters and operations names automatically. The operation is an abstractive procedure that implements the relationship of computability between two subsets of the entire set of SD parameters. Other words, this relationship allows calculating the values of the first subset of the parameters (operation output) according to known values of the second subset (operation input) if logical constraints on these parameters are satisfiable. These parameters and operations are subdivided into primitive (basis) and complex (composite) ones. Basis operations are implemented by correspondent subprograms. Basis parameters are the actual parameters of these subprograms.

For the representation of production knowledge, relations of the form $Pr: L \rightarrow O$ are introduced. The logic parameter of the subset $L \subset P$ defines the condition of the execution of the operation O . Production systems satisfy the modularity principle and ensure the dynamic processes of construction and modification of technological schemes of computational experiments. On the base of the proposed approach, it is possible to group SD objects (operations and parameters) into over-layer knowledge units called processors (C). Processors are structurally isolated and complexly organized entireties of knowledge. These SD objects represent, for example, subsystems of constructing a mathematical model or research methods similar in parameters and operations. Processors have a set of built-in objects that can be used for the research problem statement (PS). The knowledge concentrated in the processor allows automating the problem solving process by providing the means for an NPS of the problem, formulated as follows: we need to calculate the values of $D (B^0)$ from the given values of $D (A^0)$ ($A^0 \subset P, B^0 \subset P$). The NPS of the problem is hereinafter denoted as $T = (A^0; B^0; D (A^0))$ (or in short as $T = (A^0; B^0)$). The group of modules required for this problem solving is assembled using this PS with the help of inference on the SD knowledge base distributed over the computational field (CF) nodes (N). As a knowledge base, a following computational model (Knowledge Base Modified, KBM) distributed over nodes is used

$$KBM = (P, D, M, N, O, C, In, Out, Opm, Opp, Pr, Cmp, Cmo, Com),$$

where P, D, M, N, O, C are finite sets of, correspondently, parameters, parameter values, modules, CF nodes, operations, and processors. The KBM has an additional set of SD objects and relationships between them in comparison with the KBE model developed by authors in [23]. The relations

$$In \subset P \times M, Out \subset M \times P, Opm \subset O \times M, Opp \subset O \times P, Pr \subset L \times O (L \subset P), Cmp \subset C \times P, Cmo \subset C \times O, Com \subset M \times N$$

define the relationships between the correspondent sets (figure 1).

The abovementioned KBM objects are mapped onto AMP data structure as follows: distributed problem solver agents (DSA) with the ‘‘Ordinary’’ and ‘‘Condition’’ modifiers, respectively, represent the simple operation and conditional one; the module corresponds to the computational module agent (CMA). An NPS is formulated using the web-interface of the problem statement agent (PSA) and saved as a template. Unique keys of parameters are represented in the parameters vocabulary of SD; the actual values of parameters are stored in the calculated databases. Fragments of the *In*, *Out*, *Opm*, *Opp* and *Pr* relationships required for DSA functioning are stored in the local KBM^{DSA} and are used to identify neighboring with which DSA interacts and the CMA agents which DSA initiates. Thus, the KBM is distributed in such a way that each DSA agent has limited knowledge of both the capabilities of other system agents and the CF topology as a whole. *Com* and *Cmp* relations are stored in the PSA agent database and are used respectively to send messages to DSA agents and to form a local dictionary of parameters if the user selects a specific processor for the PS.

3.2. Multiagent control

Three types of agents constitute the controlling multiagent system (MAS): PSA, DSA, and CMA according to the three control levels (figure 2). At the first control level, PSA agents receive the user’s NPS *T* and send parameters names given by the sets A^0 and B^0 (signs of computability) to DSA agents, which are built-in in the selected processor. These built-in initial and goal DSA agents are created for sending out parameters from A^0 and accumulating parameters from B^0 correspondently.

At the second level, DSA agents verify the possibility of problem solving (solving-modules availability) in the computational network deployed at the CF nodes. This is stage of forming the active group of agents, step 1 in figure 2. During this verification based on self-organization mechanisms and decentralized control, DSA agents form an active group for solving the specified problem. If the problem can be solved the DSA-goal agent sends ‘‘Solvable’’ to PSA agent, the PSA agent sends input parameters values $D(A^0)$ for solving the problem (finding $D(B^0)$) (the problem solving stage, step 2 in figure 2).

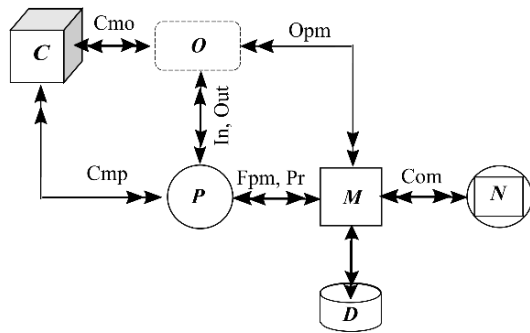


Figure 1. Conceptual scheme of SD.

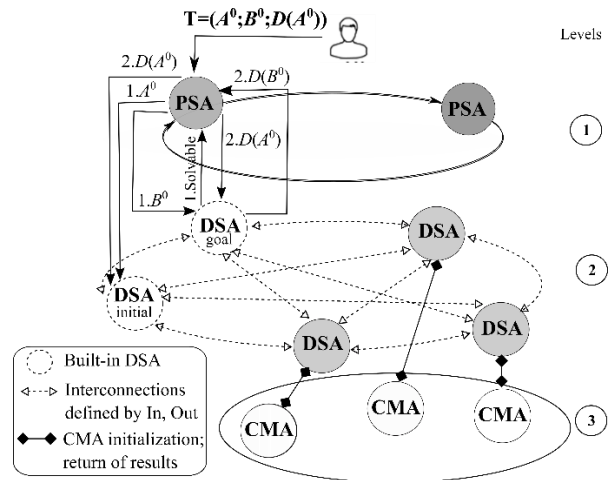


Figure 2. Control levels in AMP.

Otherwise, the user receives a notification about the impossibility of solving the problem. The method of coordinating the DSA agents is based on asynchronous behavior ‘‘on readiness of input data’’ (event-driven control) and used during both the group forming and problem solving. Direct agent interactions are defined by fragments of the *In* and *Out* relationships stored in local DSA.

The third control level is involved only for the stage of problem solving. DSA agent initiates correspondent CMA agent according to input data readiness only if the logic parameter defined by the relation Pr is TRUE. CMA agent controls the program module launch, monitoring of its executing, and sending results to DSA agent.

3.3. Conditional control structure

Let $L = \{l_1, l_2, \dots, l_q\}$, $L \subset P$ is the set of q logic parameters for controlling the launch of modules from M . The relation $Pr \subset L \times P$ will be called the conditional control structure of the KBM (similar to [23]). The semantics of this relationship is as follows: if the pair (l^*, m^*) ($l^* \in L$, $m^* \in M$) belongs to Pr , then the launch of the module m^* can be done if the value of the parameter l^* is TRUE. The parameters from L are calculated during the problem solving or sets by the user (by including in the set A^0) at the stage of the NPS T .

3.4. Multivariate calculations

Let $V = \{v_1, v_2, \dots, v_r\}$, $V \subset A^0$ be the set of r variable parameters. The necessity is observed to solve the problem T for a set of values from domain $D(V)$ repeatedly in many cases. The KBM model provides a single stage of forming an active group of agents, but the problem solving stage is repeated according to the number of variants from the domain $D(V)$. It is assumed that variants generation process is regular (algorithmic) and ends if the set of variants has been exhausted or the current one has a solution of problem T . In both cases, all active agents involved in the problem solving using multivariate calculations are deactivated. A built-in processor's operation generates the variants according to the NPS $T = (A^0 = \{V, V^{max}, V^{min}, H, \dots\}; B^0)$. The V , V^{max} , V^{min} , and H parameters are composite that include specifications of variable type and a way to increase step (for example, arithmetic or geometric progression). To organize dynamic multivariate calculations when the size of the domain D and the step of changing the variable parameters are unknown, the DSA agent is used with the "Variant" modifier, which is set when creating the agent. The DSA agent of this type iteratively performs the following sequence of actions: launching a CMA agent, sending the result to DSA neighbouring agents (according to the In , Out relationship in the local knowledge base) until a message is received from the CMA agent that no more solutions are found.

3.5. DSA agent behavior model

The behavior of the DSA agent is described by the discrete-event finite-state model FFSMwVW (Flowed Finite State Machine with Variables and Works), which is a further development of the model represented in [23, 24]. The modified model provides processing of the heterogeneous tasks flow in multivariate computations, and has the following form:

$$FFSMwVW = (\Sigma, S, \delta, x, y, G, E, W, (s_0, y_0), s_m),$$

where

- Σ and S are finite sets correspondingly of events and states;
- $\delta: \Sigma \times G \times x \times y \times S \rightarrow S \times y$ is the transition function;
- x and y are the vectors of binary correspondingly input and output;
- G and E are sets of predicates correspondingly for input variables and events;
- W is a set of program works performed by the agent;
- s_0 and s_m are the initial and final states;
- y_0 is the value of output vector y in the initial state s_0 .

When agents interact, each message with the number i initiates the formation of an event corresponding to this message, and the value of the event predicate e_i becomes TRUE. The list of events is represented in figure 3. Messages are divided into external and internal ones and processed in the order they are taken from the message queue. The transition graph of the FFSMwVW based DSA agent is shown in figure 3. The DSA agent can be in seven states. Predicates g_1, g_2, \dots, g_9 , belonging to the set G , define the condition for the transition by arcs depending on the values of the components of the input vector x and are calculated by the formulas:

$$g_1 = \overline{x_1 \cdot x_2 \cdot x_3}; g_2 = \overline{x_1 \cdot x_2 \cdot x_3}; g_3 = \overline{x_1 \cdot x_2 \cdot x_3}; g_4 = \overline{x_1 \cdot x_3};$$

$$g_5 = \overline{x_1 \cdot x_2}; g_6 = \overline{x_1 \cdot x_2}; g_7 = \overline{x_1 \cdot x_2}; g_8 = \overline{x_1 \cdot x_2}; g_9 = \overline{x_1 \cdot x_3}.$$

3.6. New DSA architecture

The new DSA agent architecture allows including this DSA in several (K_{Ag}) active groups $Ag^*_1, Ag^*_2, \dots, Ag^*_{K_{Ag}}$. In contrast to [23, 24], firstly, the data stored in the memory of the DSA agent is represented by arrays of lists instead of one-dimensional structures. Secondly, along with processing of homogeneous flows, the processing heterogeneous subtask flows, which are generated for multivariate calculations for different problems solved concurrently, is provided. Also, the structure of the transmitted messages is changed accordingly. A message router is added to the executable part of the DSA agent.

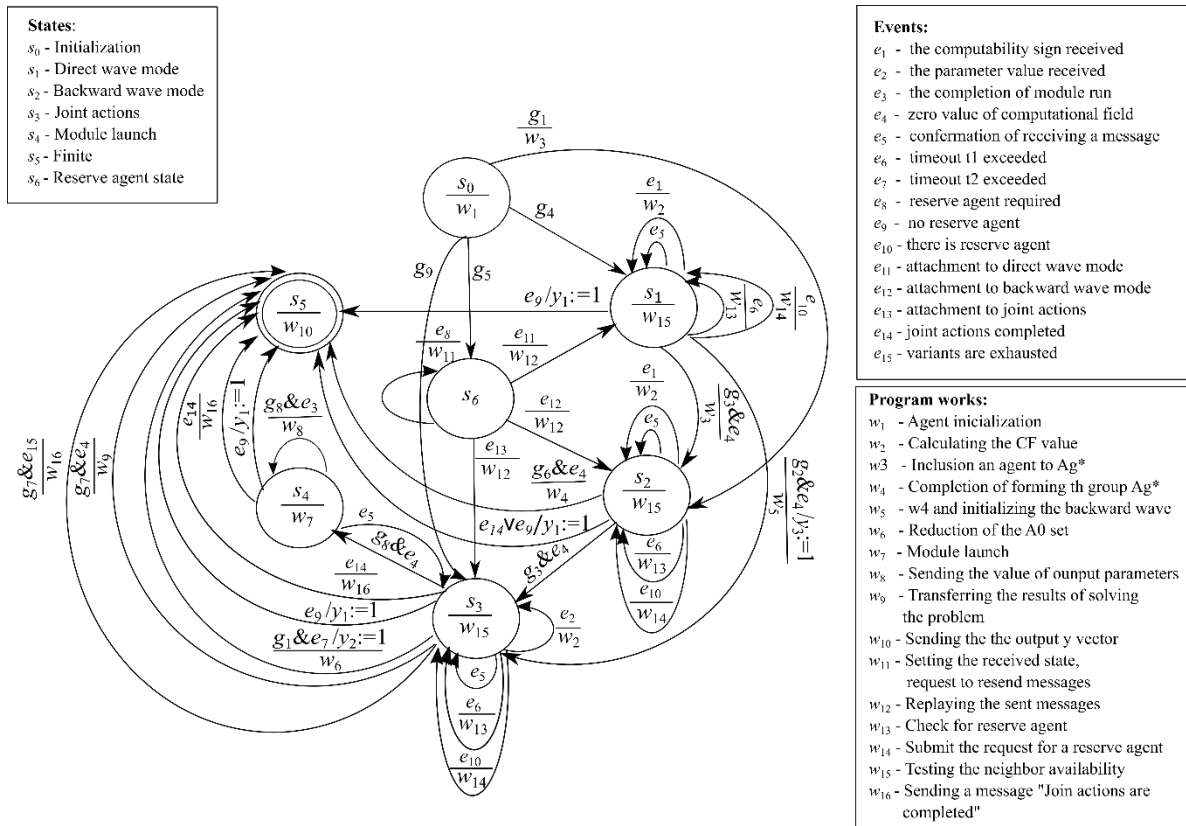


Figure 3. The transition graph of FFMSwVW based DSA agent.

3.7. AMP implementation platform novelty

The new version of previously developed by the authors the HPCSOMAS framework [7], namely HPCSOMAS-MS platform, is designed to automate the creating, deploying, testing, and usage of AMP. This version simplifies the using of cloud-based computing resources and automates the constructing AMP intended for integrated cloud and on-premises calculations. Thus, the components for deploying and updating microservices, synchronization of data installed on cloud and on-premises resources extend this version's features in comparison with [7]. The generalized scheme of constructing AMP based on HPCSOMAS-MS is presented in figure 4. The main components of HPCSOMAS-MS are the web-constructor of the AMP including subsystems for editing the SD vocabulary, web services for building and configuring of agents, and PSA structurizer and configurator; deployment wizard for microservices, and synchronizer for knowledge bases and databases.

The last four subsystems were absent in previous versions. These subsystems were developed and incorporated into the HPCSOMAS-MS platform in connection with the transition to AMP-based computing. With the help of the structurizer, computational microservices are distributed among the processors C included in the PSA agent structure. During structuring, relations Cmp and Cmo are formed. The information about which is entered into the local KBM^{PSA} . This database also stores information about users connected to the HPCSOMAS-MS system, active agent groups Ag^* formed for different problem statements (AGPS), user data update tables (UDUT), and developer data update tables (DDUT), in which monitoring results of computations are stored. The AGPS stores the set of microservices included in the NPS. UDUT is used for synchronization. DDUT and AGPS are used for automated deployment, updating, and testing of microservices. For example, in case of updating microservice, testing is performed not for all AGPS but only for those to which it belongs.

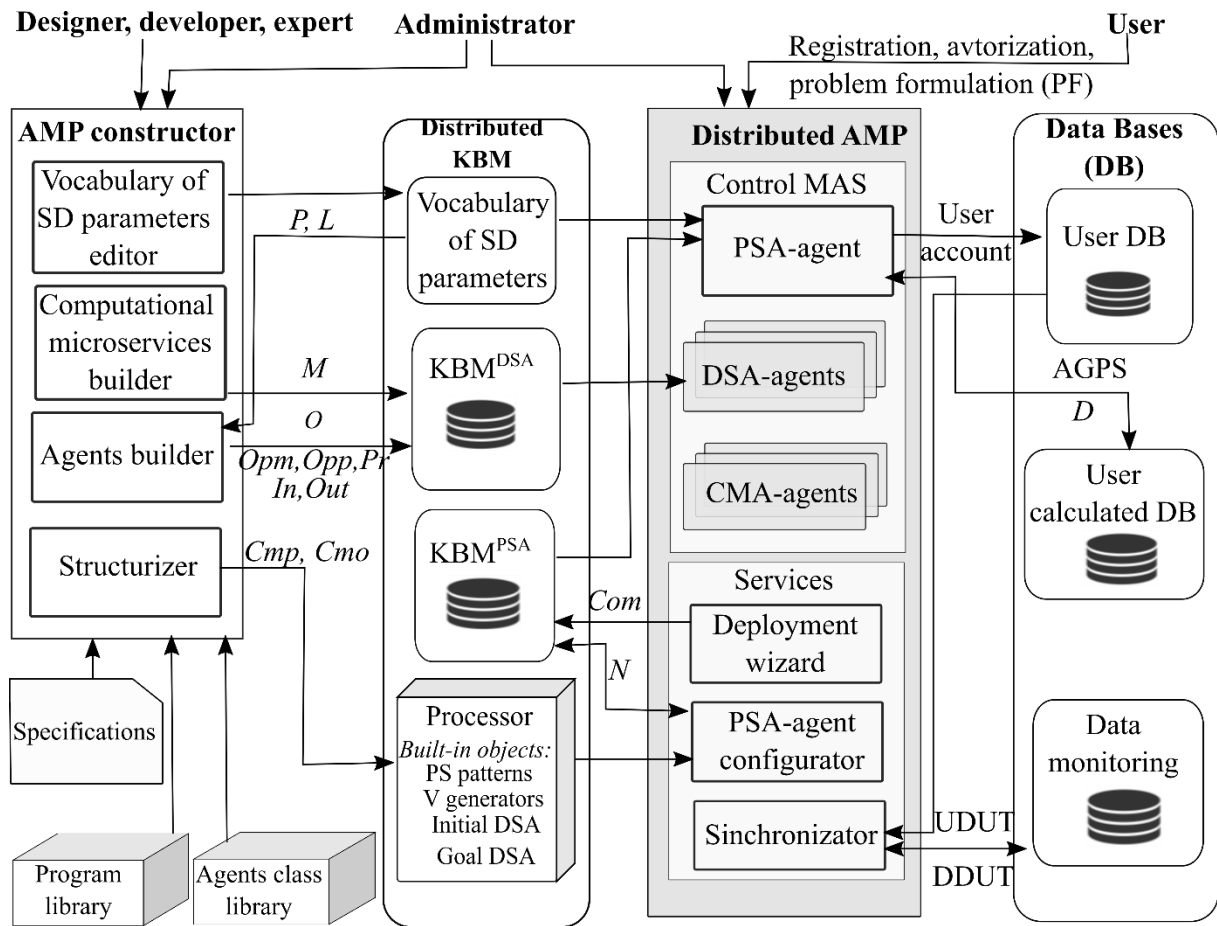


Figure 4. HPCSOMAS-MS based AMP construction.

4. Example of using AMP based on the proposed technology

The created AMP package is focused on solving problems of qualitative analysis and the structural-parametric synthesis of BDS using the author's logical method [4]. Based on this method, the solving of these problems reduce to verifying of the satisfiability of Boolean constraints on the trajectories behavior. For BDS, whose functioning is considered on a finite time interval, these constraints are written in the language of Boolean equations or Boolean formulas with quantifiers. In the first case solving the SAT problem is required. The second is related to evaluating the truthfulness of the QBF formula. At the initial stage, a Boolean model (BM) of the dynamical property of BDS is built. The structure of the BM is determined by the logical specification of the dynamical property and BDS dynamics description. At the next stage, the satisfiability of this property is verified using efficient

parallel SAT and QBF solvers. AMP includes two processors intended for a parametric synthesis of stabilizing feedback and a qualitative study of the BDS trajectories behavior on a finite time interval. As an example of the AMP use, we consider the qualitative study problems solving in the second processor. Specific parameters (a step and bounds of changing domains values) and operations (variants generators) are included into this processor for the multivariate calculations.

4.1. The problem of analyzing the structure of the state space of nonlinear BDS

Let us demonstrate the proposed approach for the practically important, and computationally complex problem of analyzing the structure of the state space for a nonlinear BDS. BDS dynamics description has the following vector-matrix form

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

where $x \in B^n$ is a state vector, $B = \{0,1\}$, n – is the dimension of the state vector; $u \in B^m$ is the input vector, m is the dimension of the input vector; $t \in T = \{1,2,\dots,k\}$ is the discrete time (number of time steps); $F(x)$ is a vector-function of the logic algebra called transition function ($F: B^n \times B^m \rightarrow B^n$). Let us define the trajectory $x(t, x^0, u(t))$ of (1) as a finite sequence x^0, x^1, \dots, x^k from the B^n set for each initial state $x^0 \in B^n$ and a finite sequence $u(t) = (u^0, u^1, \dots, u^{k-1})$ of the states of the input vector ($u^t \in B^m, t = 0,1,2,\dots,k-1$).

Qualitative analysis of BDS with inputs, based on the study of the structure of the state space, suggests solving of following problems: the searching of equilibrium states and closed trajectories (cycles), verifying of their isolation and the reachability property of the goal states set $X^* \subset B^n$ from the initial one $X^0 \subset B^n$, determining of immediate predecessors for given states. Currently, these problems are among the most important [25] for studying the dynamics of the behavior of gene regulatory networks represented by a discrete in time and state model (1), for analyzing the different stability types of a shift register with nonlinear feedback [26], and some other applications.

The model (1) is substantially nonlinear. So, existing methods in the theory of linear BDS are not applicable for (1). Therefore, we formulate the mentioned above problems of analyzing the structure of the state space of the model (1) as problems of the Boolean satisfiability [4]. The model (1) is equivalent to a single Boolean equation of the form

$$\Phi_k(x^0, x^1, \dots, x^k, u^0, u^1, \dots, u^{k-1}) = \bigvee_{i=1}^k \bigvee_{j=1}^n (x_i^t \oplus F_j(x^{t-1}, u^{t-1})) = 0, \quad (2)$$

where x_i^t and F_j are i -th components of vectors x^t and F ; \oplus denotes modulo-2 addition. For one-step transition ($k = 1$), the equation (2) takes the form

$$\Phi_1(x^0, x^1, u^0) = \bigvee_{i=1}^n (x_i^1 \oplus F_i(x^0, u^0)) = 0. \quad (3)$$

Taking into account (3), the equation (2) is

$$\Phi_k(x^0, x^1, \dots, x^k, u^0, u^1, \dots, u^{k-1}) = \bigvee_{i=1}^k \Phi_1(x^{t-1}, x^t, u^{t-1}) = 0.$$

4.2. Boolean equations for analyzing the structure of the state space

Let the input sequence $u(t)$ is constant on the all interval of functioning the system (1) functioning interval: $u(t) = (u^0, u^1 = u^0, \dots, u^{k-1} = u^0) = u^0(t)$, $u^0 \in B^m$. Then the maximum number of structures in the state space of (1) is equal to 2^m , each of which represents the autonomous behavior of (1) with a corresponding constant input action. According to [4], the required Boolean equations for a qualitative study of the state space structure of autonomous BDS on a finite time interval T are as follows:

- All immediate predecessors x^0 of the state $s \in B^n$ for given $u^0 \in B^m$ are the solutions of the following equation

$$\Phi_1(x^0, x^1, u^0(t))\Big|_{x^1=s} = 0. \quad (4)$$

- Equilibrium states for given $u^0 \in B^m$ are the solutions of the next equation

$$\Phi_1(x^0, x^1, u^0(t))\Big|_{x^1=x^0} = 0. \quad (5)$$

- The property of reachability of the goal set X^* from the set X^0 for $t \leq k$ time steps is satisfied when there is no solution for the equation

$$G^0(x^0) \vee \Phi_k(x^0, x^1, \dots, x^k, u^0(t)) \vee (\bigvee_{t=1}^k \overline{G^*(x^t)}) = 0. \quad (6)$$

Here, the equation $G^0(x^0) = 0$ defines the set of initial states, $\overline{G^*(x)}$ are the characteristic function of the set X^* .

4.3. Computational model for analyzing the structure of the state space

According to equations (4)-(6), the sets $\{\Phi_1, X^0\}$, $\{\Phi_1, S\}$, $\{\Phi_1, k, X^0, X^*\}$ are the structural elements of the BM for, correspondently, searching equilibrium states (*BMES*), searching the immediate predecessors (*BMIP*), verifying the reachability property (*BMR*). The construction of *BMES*, *BMIP*, *BMR* models (AMP parameters) are performed by *BBMES*, *BBMIP*, *BBMR* microservices (AMP operations). Microservices *EqST*, *ImPred*, and *Reach* are used for solving analysis problems for these models. The found sets of equilibrium states and immediate predecessors are stored respectively in the parameters S and A_IP . The *JA_IP* inserts the next element *IP* found by the *BBMIP* into the A_IP array if logical parameter $Y_{IP}=\text{TRUE}$. A fragment of the SD computational model is shown in figure 5.

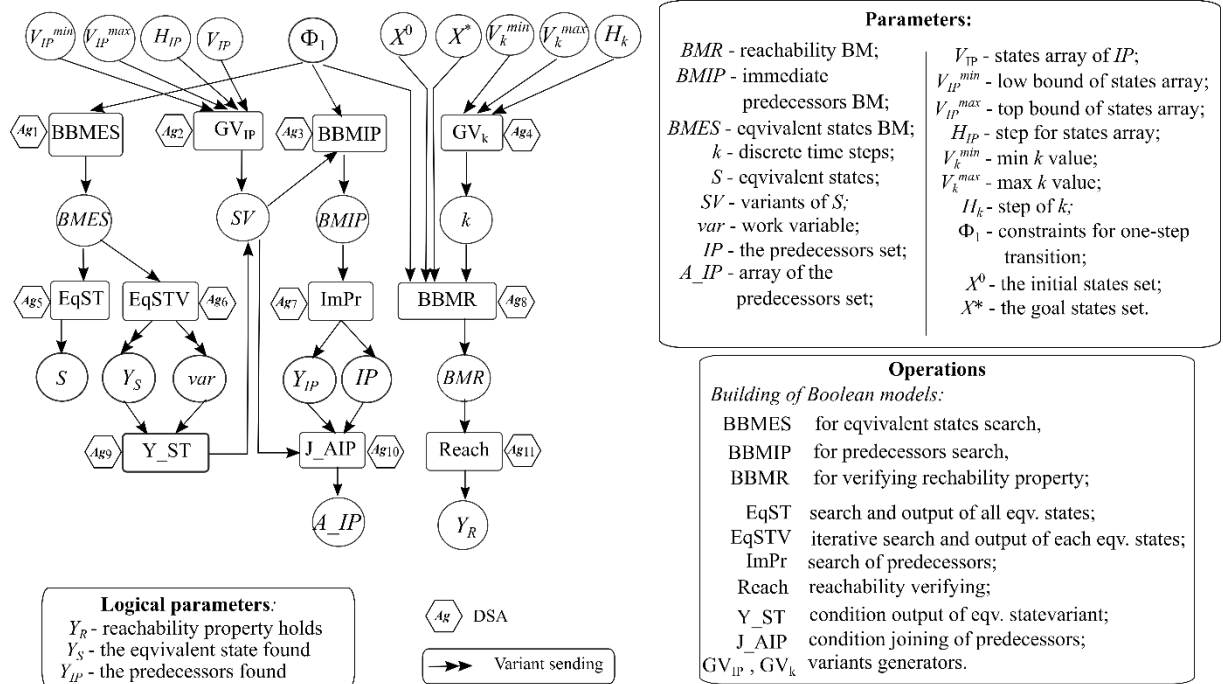


Figure 5. Fragment of computational model for analyzing the structure of the state space problem.

This fragment illustrates static multivariate computations for verifying the reachability property by the *Reach* microservice for different variants of k values (BM for k -step transitions are formed). Dynamic ones are also provided – the equilibrium states are found iteratively one at a time by the *EqSTV* microservice. The next found equilibrium state is assigned to the working variable *var* by the *EqSTV* microservice, and the logical variable Y_S is assigned TRUE. In this case, the *Y_ST* microservice convert *var* to *SV* and transmit it to *BBMIP* microservice for the verification of the

presence of immediate predecessors (an example of the pipeline: BBMES→EqSTV→Y_ST→BBMIP→ImPr→J_IP). Concurrently with the search for the next equilibrium state, the presence of predecessors (verifying the isolation property) of the previous equilibrium state is checked. In dynamic multivariate computations, pipeline computations can be performed based on functional parallelism.

Parallelism also can arise when performing multivariate heterogeneous tasks in CF using the same DSA agent in different Ag^* group. The DSA agent has a message queue, the format of which includes a task identifier and a variant identifier. Therefore, this agent can switch from one task to another if it is in a state of waiting for input data from the first task, and the data for the second one is ready. The CMA agent has a task queue. DSA agent (having "parallel" modifier when creating) duplicates the CMA one if its tasks are queuing for parallel computations in case of presence reserve resource.

4.4. Analyzing the structure of the state space

Let us consider the technology of applying Boolean constraints (4)-(6) and NPSs for studying the stability property of a nonlinear feedback shift register (NFSR) [26]. NFSR dynamics equations of closed-loop system are:

$$\begin{aligned}
x_1^t &= \overline{x_1^{t-1}} \cdot x_2^{t-1} \cdot x_4^{t-1} \cdot x_5^{t-1} \vee x_1^{t-1} \cdot \overline{x_2^{t-1}} \cdot x_4^{t-1} \cdot x_5^{t-1} \vee x_1^{t-1} \cdot x_2^{t-1} \cdot \overline{x_4^{t-1}} \cdot x_5^{t-1} \vee x_1^{t-1} \cdot x_2^{t-1} \cdot x_4^{t-1} \cdot \overline{x_5^{t-1}} \vee u^{t-1} \cdot (\overline{x_1^{t-1}} \cdot x_4^{t-1} \vee \overline{x_1^{t-1}} \cdot \overline{x_5^{t-1}} \vee \\
&\vee \overline{x_2^{t-1}} \cdot x_4^{t-1} \vee \overline{x_2^{t-1}} \cdot x_5^{t-1} \vee x_1^{t-1} \cdot \overline{x_5^{t-1}} \vee \overline{x_1^{t-1}} \cdot x_2^{t-1}) \vee u^{t-1} \cdot (x_1^{t-1} \cdot x_2^{t-1} \cdot x_4^{t-1} \vee x_1^{t-1} \cdot x_2^{t-1} \cdot x_5^{t-1} \vee x_1^{t-1} \cdot x_4^{t-1} \cdot \\
&\cdot x_5^{t-1} \vee x_2^{t-1} \cdot x_4^{t-1} \cdot x_5^{t-1}), \\
x_2^t &= x_1^{t-1} \cdot \overline{x_2^{t-1}} \cdot \overline{x_5^{t-1}} \vee u^{t-1} \cdot (x_1^{t-1} \cdot \overline{x_2^{t-1}} \cdot x_4^{t-1} \vee x_1^{t-1} \cdot x_4^{t-1} \cdot \overline{x_5^{t-1}} \vee x_1^{t-1} \cdot x_2^{t-1} \cdot x_4^{t-1} \cdot x_5^{t-1}) \vee u^{t-1} \cdot (x_1^{t-1} \cdot \\
&\cdot \overline{x_2^{t-1}} \cdot \overline{x_4^{t-1}} \vee x_1^{t-1} \cdot \overline{x_4^{t-1}} \cdot x_5^{t-1} \vee \overline{x_1^{t-1}} \cdot x_2^{t-1} \cdot x_4^{t-1} \cdot x_5^{t-1}), \\
x_3^t &= \overline{x_1^{t-1}} \cdot x_2^{t-1} \cdot \overline{x_3^{t-1}} \vee u^{t-1} \cdot (\overline{x_1^{t-1}} \cdot x_2^{t-1} \cdot x_4^{t-1} \vee x_2^{t-1} \cdot x_4^{t-1} \cdot \overline{x_5^{t-1}} \vee x_1^{t-1} \cdot \overline{x_2^{t-1}} \cdot \overline{x_4^{t-1}} \cdot x_5^{t-1}) \vee u^{t-1} \cdot (\overline{x_1^{t-1}} \cdot \\
&\cdot x_2^{t-1} \cdot \overline{x_4^{t-1}} \vee x_2^{t-1} \cdot \overline{x_4^{t-1}} \cdot x_5^{t-1} \vee x_1^{t-1} \cdot \overline{x_2^{t-1}} \cdot x_4^{t-1} \cdot x_5^{t-1}), \\
x_4^t &= x_3^{t-1}, \\
x_5^t &= x_4^{t-1}.
\end{aligned}$$

Here, $x = col(x_1, x_2, x_3, x_4, x_5)$ is the state vector, $u(t)$ is scalar input vector (syndrome [26]). Two properties of the shift register stability are considered in [26]:

- Property 1 ($u(t) = 0$, register self-control mode). The non-linear feedback shift register is stable if for each state $x \in B^n$ there exists such t that $x(t, x^0, u^0(t)=0) = 0$. Otherwise, this register is unstable.
- Property 2 (the input of the register receives an arbitrary input action $u(t)$). A shift register with nonlinear feedback is stable with respect to the input action $u(t)$ if and only if for any state x^* that this register can reach from the zero state with some input action $u(t)$, there exists such t that $x(t, x^*, u^0(t) = 0) = 0$.

Let us describe the study of analyzing the structure of the state space of the shift register. The binary register state is represented in decimal notation (x^1 is the left bit of the binary set $b_1b_2b_3b_4b_5$ and is calculated as $b_12^0+b_22^1+b_32^2+b_42^3+b_52^4$).

NPSs for computations are:

- $T_1 = (A^0=\{\Phi_1\}; B^0=\{A_IP\}, D(A^0)=\{\Phi_1=\text{"F1.cnf"}\})$ for searching immediate predecessors of equilibrium states (parallel pipeline in multivariate computations).
- $T_2 = (A^0=\{\Phi_1, V_{IP}, V_{IP}^{\min}, V_{IP}^{\max}, H_{IP}\}; B^0=\{IP\}; D(A^0)=\{V_{IP}=\{14, 30\}, V_{IP}^{\min}=1, V_{IP}^{\max}=3, H_{IP}=+1, \Phi_1=\text{"F1.cnf"}\})$ for searching immediate predecessors of each state from V_{IP} set of states (parallel pipeline in static multivariate computations).
- $T_3 = (A^0=\{\Phi_1, SV\}; B^0=\{IP\}; D(A^0)=\{SV=29, \Phi_1=\text{"F2.cnf"}\})$ for searching immediate predecessors of a particular state, for example, 29 in this case.

- $T_4 = (A^0 = \{\Phi_1, X^0, X^*, V_k^{\min}, V_k^{\max}, H_k\}; B^0 = \{Y_R\}; D(A^0) = \{V_k^{\min} = 1, V_k^{\max} = 8, H_k = *2, \Phi_1 = \text{"F2.cnf"}\}, X^0 = B^5 \setminus \{29, 14, 30, 7\}, X^* = \{0\})$ for static multivariate computations. Verifying the reachability of the set X^* from the set X^0 for different values of the level k .

Here, dynamics equations are described in files “F1.cnf” for $u^0(t) = 0$, and in “F2.cnf” for $u^0(t) = 1$. The example of the parallel pipeline when performing the NPS T_1 in dynamic multivariate computations is shown in figure 6. Functional parallelism is implemented by executing different microservices controlled by correspondent DSA at the same time interval t_i . When executing different NPS at the same time, the DSA included in both NPSs, switches between waiting mode and performing tasks for each NPS if its data is ready. The DSAs Ag_6 (variants generation) and Ag_{10} (collecting results) can be installed on the same CF node.

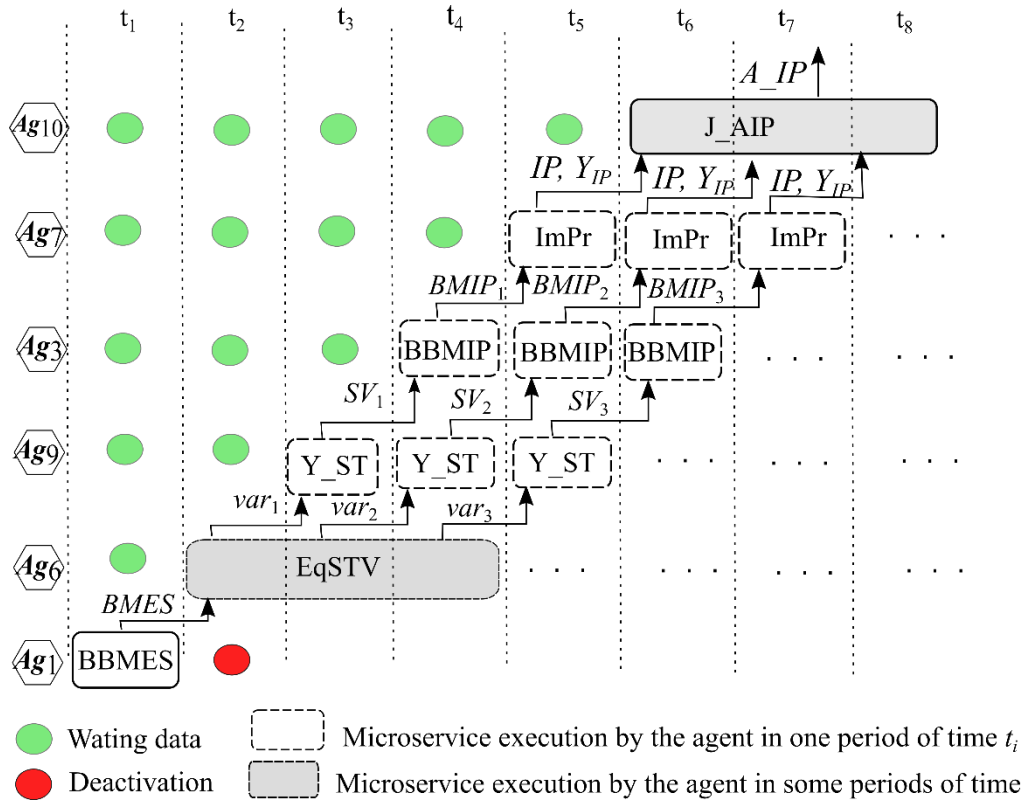


Figure 6. The parallel pipeline in dynamic multivariant computations.

The names of the input (A^0) and output (B^0) parameters are marked in the parameter list in the PSA web-interface. The value of the input data $D(A^0)$ is set in the corresponding input fields. The following explains the logic of studying stability property of shift register using the above NPSs.

The equilibrium states 0 and 29 are found by solving the equation (5) for $u^0(t) = 0$. When solving equation (4) for $u^0(t) = 0$ for these states (T_1 NPS), it turns out that state 29 is isolated (no predecessors). So the state 0 is unreachable from the state 29. Therefore, taking into account property 1, the considered shift register with nonlinear feedback is unstable.

For checking the property 2, the Boolean equation (4) is solved for $x^1 = 29$ and $u^0(t) = 1$ (T_3 NPS). As a result, the immediate predecessors of the state 29 are states 14 and 30. This equation is solved once more for these states (T_2 NPS for $x^1 = 14$ and $x^1 = 30$) for $u^0(t) = 0$. So, the state 14 has the predecessor 7, but the state 30 has none. When solving (4) analogically for $x^1 = 7$ (T_3 NPS), the predecessor is not found also. Finally, the absence of predecessors for these two states (30 and 7) both for $u^0(t) = 0$ and $u^0(t) = 1$ means unreachability of 29 for any input $u(t)$. Therefore the reachability set is $X^0 = B^5 \setminus \{29, 14, 30, 7\}$.

The equation (6) is solved for checking the reachability property of the set $X^* = \{0\}$ for $k = 1, 2, 4, 8$ (T_4 NPS). Elements of sets X^0 and X^* are the roots of the following Boolean equations (state 0 is excluded from X^0):

$$G^0(x): \overline{x_5} \cdot \overline{x_4} \cdot \overline{x_3} \cdot x_2 \cdot x_1 \vee \overline{x_5} \cdot \overline{x_4} \cdot \overline{x_3} \cdot x_2 \cdot x_1 \vee \overline{x_5} \cdot \overline{x_4} \cdot \overline{x_3} \cdot x_2 \cdot x_1 \vee \overline{x_5} \cdot \overline{x_4} \cdot \overline{x_3} \cdot x_2 \cdot x_1 \vee \overline{x_5} \cdot \overline{x_4} \cdot \overline{x_3} \cdot x_2 \cdot x_1 \vee \overline{x_5} \cdot \overline{x_4} \cdot \overline{x_3} \cdot x_2 \cdot x_1 = 0,$$

$$G^*(x): x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 = 0.$$

As a result, for $k = 8$ the reachability property is satisfied for X^* . Therefore, taking into account

$$u^0(t)=0$$

$$u^0(t)=1$$

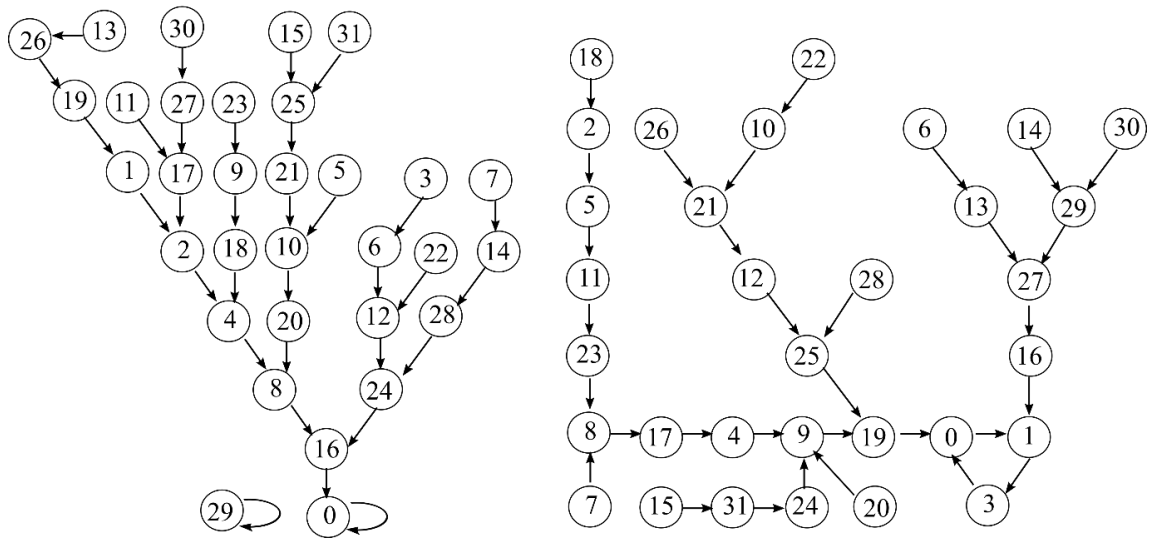


Figure 7. The state diagrams of the shift register.

Property 2, a register with nonlinear feedback is stable relative to the input action $u(t)$. State transition diagrams (figure 7) allows verifying the stability property of the shift register.

This example illustrates using the NPS for research problem solving, and the static/dynamic multivariate calculations conducted by active agents group for analyzing the structure of the state space of shift register.

5. Conclusion

We develop the automation technology for creating, deploying, and supporting the AMP functioning in a hybrid computing environment. This technology provides both functional parallelism and data parallelism at dedicated CF nodes for the semantic network of agents. This functionality is based on the AMP microservices architecture, the means of AMP computational model for conditional module launch, and multivariate computations, decentralized multiagent control, self-organization mechanisms, and DSA agent architecture based on FFSMwVW model.

Using this technology, we developed AMP to automate a qualitative study of binary dynamic systems based on the Boolean constraints method and demonstrated its application to solve the problem of analyzing the structure of the states space of the shift register and its stability.

The developed AMP can be used to solve a wide range of tasks, including the study of the dynamics of gene regulatory networks represented by a discrete both in time and a state, model.

Acknowledgments

This work was supported by 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. 7, project “Methods, algorithms and tools for the decentralized group solving of problems

in computing and control systems” ” (reg. no. AAAA-A18-118031590006-2). The authors would like to thank the Irkutsk Supercomputer Center of SB RAS for providing access to cluster computational resources.

References

- [1] Mazzara M, Khanda K, Mustafin R, Rivera V, Safina L and Sillitti A 2018 Microservices science and engineering ed P Ciancarini, S Litvinov et al *Proc. of the 5th International Conference in Software Engineering for Defence Applications. SEDA. Advances in Intelligent Systems and Computing* (Cham: Springer) 717 pp 11-20
- [2] Netto M A, Calheiros R N, Rodrigues E R, Cunha R L and Buyya R 2018 HPC cloud for scientific and business applications: taxonomy, vision, and research challenges *ACM Comput. Surv.* **51** 8:1-8:29
- [3] Somov Ye I, Butyrin S, Oparin G A and Bogdanova V G 2016 Methods and software for computer-aided design of the spacecraft guidance, navigation and control systems *MESA* **7**(4) 613-624
- [4] 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)
- [5] Newman S 2015 *Building Microservices* (O'Reilly)
- [6] Fisher C 2018 Cloud versus on-premise computing *American Journal of Industrial and Business Management* **8** 1991-2006
- [7] 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 (Cham: Springer, LNCS) **10421** pp 278-288
- [8] Oberhauser R 2017 Microflows: Enabling agile business process modeling to orchestrate semantically-annotated microservices ed R Oberhauser and S Stigler *Seventh Int. Symp. on Business Modeling and Software Design* (SCITEPRESS) **1** pp 19-28
- [9] Ghani A and Zakaria M 2018 Method for designing scalable microservice-based application systematically: a case study *IJACSA* **9**(8) 125-135
- [10] 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
- [11] Rodrigues N, Leitão P and Oliveira E 2014 Dynamic composition of service oriented multi-agent system in self-organized environments *Proc. of the 2014 Workshop on Intelligent Agents and Technologies for Socially Interconnected Systems* pp 1-6
- [12] Collier R W, O'Neill E, Lillis D and O'Hare G 2019 MAMS: Multi-Agent MicroServices *Proc. The WWW Conf.* pp 655-662
- [13] Mayer P, Velasco J, Klarl A, Hennicker R, Puviani M, Tiezzi F, Pugliese R, Keznikl J and Bureš T 2015 The autonomic cloud ed M Wirsing and M Hözl *Software Engineering for Collective Autonomic Systems* (Cham: Springer, LNCS) **8998** 495-512
- [14] Hennicker R and Klarl A 2014 Foundations for ensemble modeling – the Helena approach ed S Iida et al (Berlin, Heidelberg: Springer, LNCS) **8373** pp 359–381
- [15] Hözl M and Wirsing M 2011 Towards a system model for ensembles ed G Agha et al *Formal Modeling: Actors, Open Systems, Biological Systems* al (Berlin, Heidelberg: Springer, LNCS) **7000** pp 241-261
- [16] Merkel D 2014 Docker: lightweight Linux containers for consistent development and deployment *Linux Journal* **2014**(239)
- [17] Florio L 2015 Decentralized self-adaptation in largescale distributed systems *Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering* (New York: ACM) pp 1022-1025
- [18] Bures T, Gerostathopoulos I, Hnetyinka P, Keznikl J, Kit M and Plasil F 2013 Deeco: An ensemble-based component system *Proc. of the 16th International ACM Sigsoft Symp. on*

- Component-based Software Engineering* (New York, USA: ACM) pp 81–90
- [19] Brueckner S and Czap H 2006 Organization, self-organization, autonomy and emergence: States and challenges *ITSA* **2**(1) 1-9
 - [20] Fernández H, Priol T and Tedeschi C 2010 Decentralized approach for execution of composite Web services using the chemical paradigm *Proc. of the 8th IEEE Int. Conf. on Web Services* (Miami: FL) pp 139-146
 - [21] Buguillo J 2018 *Self-organizing Coalitions for Managing Complexity* (Springer International Publishing) **29** pp 89-100
 - [22] Krivic P, Skocir P, Kusek M and Jezic G 2018 Microservices as agents in IoT systems *Int. symp. Agent and Multi-Agent Systems: Technology and Applications. KES-AMSTA 2017. Smart Innovation, Systems and Technologies* ed G Jezic and M Kusek (Cham: Springer) **74** pp 22-31
 - [23] Bychkov I V, Oparin GA, 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)
 - [24] Oparin G A, Bogdanova V G, Pashinin A A and Gorsky S A Distributed solvers of applied problems based on microservices and agent networks *Proc. of the 41st Int. Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, 2018*. P. 1415-1420
 - [25] Akutsu T, Hayashida M and Tamura T 2008 Algorithms for inference, analysis and control of Boolean networks *Proc. of the Int. Conf. on Algebraic Biology* pp 1-15
 - [26] Massey J L and Li R W 1964 Application of Lyapunov's direct method to error-propagation effect convolutional code *IEEE Trans. IT* **10**(3) 248-250