Reconfigurable monitoring for telecommunication networks

Man Tianxing [Corresp., 1] , **Vasiliy Yurievich Osipov** [2] , **Alexander Ivanovich Vodyaho** [3] , **Andrey Kalmatskiy** [4] , **Natalia Alexandrovna Zhukova** [2] , **Sergey Vyacheslavovich Lebedev** [3] , **Yulia Alexandrovna Shichkina** [3]

[1] ITMO University, Saint Petersburg, Saint Petersburg, Russia

[2] Saint-Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences, Saint Petersburg, Saint Petersburg, Russia

[3] St. Petersburg State Electrotechnical University, Saint Petersburg, Saint Petersburg, Russia

[4] Google, New York, America

Corresponding Author: Man Tianxing
Email address: mantx626@gmail.com

This article addresses the monitoring problem of the telecommunication networks. We consider these networks as multilevel dynamic objects. It shows that reconfigurable systems are necessary for their monitoring process in real life. We implement the reconfiguration abilities of the systems through the synthesis of monitoring programs and their execution in the monitoring systems and on the end-user devices. This article presents a new method for the synthesis of monitoring programs and develops a new language to describe the monitoring programs. The programs are translated into binary format and executed by the virtual machines installed on the elements of the networks. We present an example of the program synthesis for real distributed networks monitoring at last.

1 **Reconfigurable monitoring for telecommunication**

2 **networks**

3   Man Tianxing[1], Vasiliy Yurievich Osipov[2], Alexander Ivanovich Vodyaho[3], Andrey Kalmatskiy[4],

4   Natalia Alexandrovna Zhukova[12], Sergey Vyacheslavovich Lebedev[3], Yulia Alexandrovna

5   Shichkina[3]

6   [1] ITMO University, Saint Petersburg, Russia

7   [2] Saint-Petersburg Institute for Informatics and Automation of the Russian Academy of

8   Sciences, Saint Petersburg, Russia

9   [3] St. Petersburg State Electrotechnical University, Saint Petersburg, Russia

10   [4] Google, New York, America

12   Corresponding Author:

13   Man Tianxing[1]

14   Kronverkskiy Prospekt, 49, St Petersburg, 197101, Russia

15   Email address: mantx626@gmail.com

17 **Abstract**

18 This article addresses the monitoring problem of the telecommunication networks. We consider
19 these networks as multilevel dynamic objects. It shows that reconfigurable systems are
20 necessary for monitoring processes in real life. We implement the reconfiguration abilities of the
21 systems through the synthesis of monitoring programs and their execution in the monitoring
22 systems and on the end-user devices. This article presents a new method for the synthesis of
23 monitoring programs and presents a new language to describe the monitoring programs. The
24 programs are translated into binary format and executed by the virtual machines installed on the
25 elements of the networks. We present an example of the program synthesis for real distributed
26 networks monitoring at last.

27 **I. Introduction**

28 Nowadays, the success of human activity in many areas largely depends on the efficiency of the
29 functioning of telecommunication networks. Telecommunications networks today are used in a
30 variety of fields of human activity. Thus, the use of radar and radar-optical data to solve a
31 massive class of problems in the field of transport, agriculture, classification of area and point
32 objects, monitoring of anthropogenic activities, and other tasks seem to be extremely promising.
33 At least a dozen of new radar satellites are planned to be launched in the coming years, all cars
34 today are equipped with radar. The intelligent processing of this data requires the use of edge
35 and cloud computing, which entails the new challenges to the telecommunication network
36 design.

37 Damage or incorrect operation of such networks can be a reason for severe losses. So,
38 it is necessary to have adequate tools and methods for maintaining such networks. It can be
39 done through permanent monitoring. Monitoring ensures safety operation, quick detection, and
40 prevention of network failures. Many factors cause changes in the structure, the state, and the
41 behavior of the networks. Thus, there is a need to reconfigure monitoring processes.
42 Reconfigurable monitoring processes should allow us to gather the data about dynamic
43 telecommunication networks and process them according to the current state of these networks
44 and estimations of their expected use.

45 In the telecommunication networks, objects of monitoring are multiple technical devices
46 of end-users. Sets of parameters characterize the state of the objects and their elements.
47 Centralized and decentralized schemes for object monitoring are used. Centralized schemes
48 assume that monitoring systems gather data from objects. Also, data can be gathered by the
49 objects and transferred to the monitoring systems. In both schemes, predefined processes are
50 commonly used. Gathering all possible data leads to a significant increase in network traffic.
51 When data is partially collected, it does not allow quickly to detect failures of the objects, identify
52 their reasons.

53 For the reconfiguration of monitoring processes, business rules can be used. The rules
54 define processes behavior for different known situations. This approach is well applicable for
55 monitoring when networks have permanent structures and permanent conditions of use.
56 The problem of monitoring of dynamic networks remains unsolved. To solve the problem
57 monitoring systems must have abilities to build and rebuild the monitoring processes
58 dynamically.

59   In the paper, we propose a new method for building monitoring processes based on
60 using the theory of synthesis. To execute the processes a new language and a new virtual
61 machine (VM) are developed.
62   The rest of the paper is structured as follows. In the second section, the capabilities of
63 the existing monitoring systems are analyzed. It is shown that for dynamic telecommunication
64 networks reconfigurable systems have not been considered till now. Then we describe
65 monitoring processes and propose new models for monitoring programs and the method for
66 their synthesis that allow develop reconfigurable monitoring systems. In the fourth section we
67 present the implementation of the proposed method, which contains the description of the
68 language for monitoring programs, a compiler for the language, and a virtual machine that is
69 capable of executing the compiled monitoring programs on end-user devices. In the fifth section,
70 a case study is given to prove the effectiveness of the method. The last section is the
71 conclusion.

II. Materials & Methods

Related work

74 Today monitoring systems are used almost in all subject areas – medical (Baig et al. 2017;
75 Mukhopadhyay 2014; Sohn et al. 2003; Yang et al. 2016, Shichkina et al. 2019), environment
76 (Gaikwad & Mistry 2015; Kumar et al. 2012), social network (Chakravarthi et al. 2015), energy
77 (Eissa et al. 2015), production (Singh et al. 2010), urban, economy (Bello et al. 2018;
78 Sakhardande et al. 2016), agriculture (Suma et al. 2017), transport (Shichkina et al. 2019),
79 fishery (Raju & Varma 2017), etc. For many subject areas, including telecommunication,
80 monitoring systems are playing a critical role.
81   In traditional monitoring systems (Zabbix, http://www.zabbix.com/), (Nagios,
82 http://www.nagios.org/), (Zenoss, https:// www.zenoss. Org/), (Pandora FMS
83 https://pandorafms.com/), (Icinga https://icinga.com/), (Sensu http://sensu.io/), (GroundWork
84 Monitor https://www.gwos.com /), data is collected and then transmitted to monitoring systems
85 in the form of the data streams. The data is processed by monitoring systems which have a
86 distributed tree structure. Each element can collect and process data from multiple objects. The
87 higher-level elements aggregate the data processing results of the lower-level elements. The
88 tree structure allows the monitoring systems to have a sufficiently high level of performance. If
89 the observed object changes and data is generated, redesign of the monitoring systems is a
90 necessary process. Systems redesign assumes active involvement of software architects and
91 programmers.
92   In the past few years, the monitoring systems have evolved significantly. Developed
93 monitoring systems are based on the integration of multiple modern technologies (Morgan &
94 O'Donnell 2018). In particular, in these systems, data is collected using Internet of Things
95 technologies (Maksymyuk et al. 2017; Myint et al. 2017; Raju & Varma 2017; Sakhardande et
96 al. 2016; Suma et al. 2017; Yang et al. 2016). Data processing employs statistical methods,
97 data mining algorithms, and machine learning algorithms (Ge et al. 2013). These monitoring
98 systems are a type of adaptive systems. They assume changes in both data gathering
99 processes and processes of data processing. Adaptive systems use flexible data processing
100 models (Cabal-Yepez et al. 2012; Dovžan et al. 2014) or business rules (Al Mamun et al. 2016;
101 Koetter & Kochanowski 2015) to build and rebuild processes. The likelihood of their ability to

102 adapt in different cases depends on the composition of the selected business rules and the
103 defined values of customizable parameters of the processing models (Singh et al. 2010).
104   The reconfigurable monitoring systems have advanced capabilities for adaptive data
105 gathering and processing. The features of reconfigurable systems are discussed in (Korableva
106 et al. 2017; Lyke et al. 2015). In monitoring systems, reconfiguration operations can be
107 considered at several levels: sensor level, embedded system level, and application level. At the
108 sensor level, reconfiguration problems are solved by developing new programmable sensors
109 (Myint et al. 2017). The new programs are uploaded for updating the logic of the sensors. For
110 the adaptation of the embedded monitoring systems, they equip with reconfigurable data
111 preprocessing modules (Cabal-Yepez et al. 2012). At the application level, the adaptability is
112 achieved through customizable interfaces (Fu 2012), usage of genetic algorithms (Thompson
113 2012), and so on. It is also possible to synthesize new configurations for the systems and their
114 components at the level of field-programmable gate arrays (Cong et al. 2011; Cong & Minkovich
115 2007; Lysecky et al. 2004; Rubin & DeHon 2011).
116   For implementing reconfigurable systems, several methods have been proposed. In
117 (Morgan & O'Donell 2014), service-oriented architectures of reconfigurable systems are
118 presented. The problems of message routing and service orchestration in such systems are
119 discussed. Technological solutions for building reconfigurable systems are given in (Silva et al.
120 2018).
121   One of the limitations of the existing systems is that they cannot reconfigure themselves
122 dynamically. For dynamic reconfiguration, they have to be able to synthesize programs for data
123 gathering and data processing and execute them. Some of the processes should be performed
124 in the monitoring systems and some of the end-user devices.
125   To date, only a few dynamically reconfigurable systems exist. One of such systems has
126 been developed for monitoring the status and performance of a supercomputer network
127 (Stefanov, K., et al. 2015). The monitoring system is considered as a set of subsystems. The
128 subsystems process data about the performance of one or more tasks. Performance metrics are
129 calculated on the fly. To this purpose, a program module is dynamically created for each
130 process that is performed to solve the general task, and the transmission paths for the
131 calculated metrics are dynamically determined. The module oriented architecture of the systems
132 and their ability to create the modules dynamically make them able to adapt to new data
133 sources and new processing methods. When creating a new module, the nodes on which it will
134 be executed are selected based on the current load on the computer network. As a result, a
135 balanced load of the nodes is ensured. The nodes of the system can process a part of the data
136 about the state of the network. With these additional resources for solving the monitoring
137 problems are obtained. The results of using these monitoring systems have shown that they
138 allow solving the tasks of monitoring of supercomputer networks.Reconfigurable systems and
139 dynamically reconfigurable systems have not yet been developed in the field of
140 telecommunications.

141 III. Multilevel Synthesis of Monitoring Programs

142 **Monitoring Objects, Processes, and Programs in Telecommunication Networks.**
143 Telecommunication networks have a multilevel structure. The elements of the higher levels are
144 groups of objects. Within the lower levels, separate objects are considered. These objects are

145   end-user devices. The components and the subcomponents inside characterize them. There
146   are multiple links between the elements at all levels.
147   Monitoring processes are oriented on providing data about the state of the end-users'
148   devices to the customer service. These devices can gather data about their state and process it.
149   Monitoring systems can interact with the devices. Within the interaction, data can be received
150   from the devices, and commands on data gathering and data processing can be sent to them.
151   The devices can perceive both separate commands and programs that contain multiple
152   commands.
153   Monitoring programs are built and rebuild by monitoring systems. New programs are built-in
154   multiple cases, such as changes in the structure, state, or behavior of the devices or definitions
155   of new requirements to the results of the monitoring. Requirements can refer to both the output
156   data about the state of the network and the conditions of the monitoring, for example,
157   admissible time modeling, accuracy and reliability of the models, consumed technical resources,
158   etc.
159   There are two primary sources of data about the state of the devices. They are the
160   messengers that are sent from the devices and the log files that they produce. The log files are
161   sequences of records that describe events registered by the devices. Related data on factors
162   affecting the network operation can be received from other sources.
163   The operators of the customer service define the requirements for the results of the
164   monitoring. Figure 1 represents the structure of the monitoring process.
165   **Figure 1: The structure of the monitoring process.**
166   According to Fig 1, monitoring processes assume gathering data about the target objects (end-
167   user devices). Current models of target objects are built using gathered data. These models
168   reflect the state of the objects based on available data. Current models are compared with the
169   required models. They are defined according to the requirements of the end-users (operators of
170   customer service). Usually, these two models do not match. Current models can have many
171   undefined parameters in comparison with the required models. If the models do not match, then
172   new monitoring programs are synthesized. Possible ways of transitions between current and
173   required models are considered to synthesize new programs. Each transition determines
174   additional elements of data that are needed to reach the target model. Thus, after constructing
175   sequences of transitions, it becomes known what data should be gathered. New monitoring
176   programs contain sequences of commands that allow gathering required data elements.
177   The formally monitoring process PR can be defined as $PR : O' \rightarrow O''$ where O' is the
178   model that reflects the state of the objects in the network. The model O' is the current model
179   that is build using gathered data. The model O'' is the model that is required by the user. The
180   monitoring processes are targeted at building current models that match the required models.
181   The synthesis of monitoring programs is an essential part of the monitoring processes.
182   Synthesized programs are defined as the complement of O'' concerning O':
183   $PRG = O'' \setminus O' = \{ e \in O'' \mid e \notin O' \}$.
184   The synthesis of the monitoring programs is ensured through the following:
185   •   The networks are commonly described in discrete time and discrete space. Thus automata
186   are used as the models of the networks. Automata describe the states of the networks.
187   Since there are too many possible states, some of them can be unknown, it is not possible

188   to describe all the states of the automata apriori. Due to that relatively finite automata are
189   used (Osipov, V. Yu. et al. 2016). They are finite only for one moment of time. At each time
190   step, all the parameters of the automata can be redefined.
191 ● Traditional methods of synthesis have high computational complexity. The complexity can
192   be reduced if we take into account the multilevel structure of the networks. To synthesize
193   models of objects with multilevel structure multilevel methods of synthesis can be used.
194   Compared to single-level methods, they allow increase the speed of the synthesis of the
195   required programs in times. It is achieved due to the fact that the number of the elements in
196   the models at upper levels is small in relation to the lower levels.
197 ● The synthesized programs can be described using a new language. It is oriented on
198   describing the programs for monitoring of telecommunication networks. The programs that
199   are written using this language can be compiled and executed on end-user devices.
200 **Automata Models for Programs Synthesis.**
201 The models of the objects are multilevel structures that represent the objects as multiple sets of
202 parameters. Sets of parameters and separate parameters are linked within separate levels and
203 between the levels.
204   We suggest using multilevel automata models to synthesize the programs of monitoring for
205 the objects that assume gathering information at different levels.
206   The model of the automata at one level can be defined in the following way (Osipov, V. Yu.
207 et al. 2016; Osipov, V. Yu. et al. 2017):
208   <AI, AO, AS, AR>, where AI - a set of admissible input data, AO - a set of admissible output
209 data, AS - a set of admissible internal states, AR - an admissible set of transition functions
210 between the elements of the set of input data and the elements of the set of output data.
211   So, the entire multilevel model is
212        $< <AI^0, AO^0, AS^0, AR^0>; <AI^1, AO^1, AS^1, AR^1>; ..... <AI^N, AO^N, AS^N, AR^N> >,$
213   where the super index defines the level of the automata model.
214   At each step of program synthesis, automata can be in one of the admissible states. Within
215 each step, the sets of admissible states and transition functions are rebuilt, taking into account
216 the achieved state and the conditions of synthesis. Thus the automata at step r is defined as:
217 $<I_r, O_r, S_r, F_r, AI_{r-1}, AO_{r-1}, AS_{r-1}, AR_{r-1}>$, where I - input data, O - output data, S - internal state, F
218 - transition functions (conditions). Parameters I, O, S, R are defined for the step r and
219 parameters AI, AO, AS, AR - for the step r-1.
220 **Method of Synthesis of Monitoring Programs.**
221 The method allows synthesizing programs for objects monitoring in conditions when the current
222 and the target models of the objects are defined. It is required to gather data about the objects
223 that are necessary to move towards required models from current models.
224   The method uses direct and backward inference. The direct inference is used to prove that
225 the transition between models is possible and that the target model exists. As a result of the
226 direct proof, multiple sequences of transitions can be defined. The backward inference allows
227 restoring programs from the results of the proof. The inference is carried out according to the
228 scheme mirrored to direct inference.
229   In the conditions, when the existence of the target models cannot be proved, the
230 possibilities of changing the models can be considered.

231   In the automata model for program synthesis, the set of input data elements $\{d_s\}$ are defined
232 according to the current model of the object, the set of output data elements $\{d_w\}$ are elements
233 of the target model of the object. Transition functions F_{zv} reflect links between elements of the
234 models: $F_{zv}(d_{zve})$ ->(d_{zva}), where e = [1, E_z] indicates elements of the models that are required to
235 achieve an element indicated as d_{zva}.

236 ***Method for building sequences of transitions between object models based on direct***
237 ***inference.***

238   Below the description of the method is given:
239   *Input data*. $\{d_s\}$ - the sets of the input data elements; $\{d_w\}$ - the sets of the output data
240 elements; F_{zv} - transition functions between the elements.
241   *Output data*. TransitionSequence - sequences of transitions that allow reach elements $\{d_w\}$
242 of the target model having elements $\{d_s\}$ of the current model.
243   The pseudo code of the method is the following:
244   Function FindAchievableElements(maxLevel, M, CurrentM, TargetM) is

```
245  1      for each l in [maxLevel..0] do
246  2        for each i in [0..M.funConditions[l].length] do
247  3         if checkCondition(M.funConditions[l][i]) == true do
248  4         elementPairs = cartesianProduct(CurrentM.Elements[l], TargetM.Elements[l])
249  5          for each j in [0..elementPairs.length]
250  6           if profTransition(M.vzLinkingFun[l][i](elementPairs[j].first, elementPairs[j].second) =true do
251  7             M.AchievedFrom[l] = add(M.AchievedFrom[l], elementPairs[j].first)
252  8             M.AchievedTo[l] = add(M.AchievedTo[l], elementPairs[j].first)
253  9             M.AchievedWith[l] = add(M.AchievedWith[l], M.vzLinkingFun[l][i])
254  10            M.AchivedWhile[l] = add(M.AchivedWhile[l], checkCondition[l][i])
255  11            remove(M.NonProven[l+1], elementPairs[j].second)
256  12          else
257  13            M.NonProven[l]= add(M.NonProven[l], elementPairs[j].second)
258  14          end
259  15       if M.NonProven[l].length != 0 && maxLevel != 0 do
260  16         FindAchievableElements(maxLevel-1, M, CurrentM, TargetM)
261  17        end
262  18      return M
```

263   Function DirectProgramSynthesis(maxLevel, M, CurrentM, TargetM) is

```
264  1     M = FindAchievableElements(maxLevel, M, CurrentM, TargetM)
265  2     if M.NonProven.length == 0 do
266  3      for each l in [0..maxLevel] do
267  4        M.TransitionSequence = zip(zip(M.AchievedWith[l], M.AchievedWhile[l]),zip(M.AchievedFrom[l],
268     M.AchievedTo[l])) /* to get a list of quads of the form (function, condition, input, output)*/
269  5       else error 'no transition is found'
270  6       return M
```

271

272 ***Method for restoring programs from the results of the proof based on backward***
273 ***inference.***

274   Below the description of the method is given.
275   *Input data*. {dw} - the sets of the input data elements; {ds} - the sets of the output data
276 elements. Target data elements are considered as the input, and the input data elements as the
277 target elements because the inference is backward.
278   *Output data*. PRG -models of monitoring programs.
279   The pseudo-code of the method is the following:
280   Function BackwrdProgramSynthesis(maxLevel, M, CurrentM, TargetM) is

```
281   1        PRG = initializeProgram
282   2        MProc = initializeProgramModel
283   3        /* Build the model of monitoring program using backward inference, TargetM and CurrentM are
284   swapped*/
285   4        MProc = DirectProgramSynthesis(maxLevel, MProc, TargetM, CurrentM)
286   5        MProg = reproduceTransitionSequences(MProc)
287   6        /* Build the model of the monitoring program according to the synthesized structure */
288   7        for each l in [(maxLevel-1)..0] do
289   8          for each i in [0.. MProc.AchievedWith[l].length] do
290   9            z = getFunKind(MProc.AchievedWith[l][i])
291   10           v = getFunType(MProc.AchievedWith[l][i])
292   11           MProg.StructureElements[l] = add(MProg.StructureElements[l], buildStructureElement(z, v))
293   12         End
294   13         for each i in [0.. MProc.AchivedWhile[l].length] do
295   14           conditions = MProc.AchievedWhile[l][i]
296   15           MProg.Condition[l] = add(MProg.Condition[l], buildConditions(conditions))
297   16         end
298   17         MProg.Program[l] = buildProgramForLevel(MProg.StructureElements[l], MProg.Condition[l])
299   18         /* transform program structure elements of each level to the resulting program structure */
300   19         PRG = buildProgram(PRG, MProg.Program[l])
301   20       end
302   21     return PRG
```

The resulting program is a sequence of program structure elements that can be converted to program code written using scripting languages.

Complexity of Multilevel Synthesis of Monitoring Programs.

The upper boundary of time T_H required for program synthesis using the proposed method can be estimated by the following formula:

$$T_H \approx c \sum_{i=0}^{K} m_i^2 \le c \left(\sum_{i=0}^{K} m_i \right)^2 \qquad (1)$$

where: c – constant-coefficient; m_i - the number of conditions on i-th level. It is necessary to mention that m_i is essentially less than the total number of the conditions of the automata model.

Taking into account that on upper levels each inference step is equivalent to n_i steps at the level "0", one can get a lower boundary of time T_L of multilevel synthesis of the programs,

$$T_L \approx c \sum_{i=0}^{K} \frac{m_i^2}{n_i^2} \le c \sum_{i=0}^{K} m_i^2 \qquad (2)$$

The average estimate of time T of multilevel synthesis concerning (1), (2) is equal to $T = (T_L + T_H)/2.$

IV. Implementation of Monitoring Programs

Language for monitoring programs.

We design the language to describe the programs for monitoring of telecommunication networks. The programs are intended for execution on end-user devices.

The following requirements were imposed on the capabilities of the language. The language should allow configurable filtering and data aggregation parameters, determine the composition of messages sent from the devices, specify commands for performing actions by the devices, and determine the conditions for their execution.

325 **Table 1: Vocabulary of the modeling language.**

326   Vocabulary and syntax are defined for the modeling language. The vocabulary is presented
327 in Table. 1, the syntax is given as follow:

328   *program=: 'module' id ';' (global_var_definition | function_definition)* <end of file>*
329   *global_var_definition=: id(name) '=' expression ';'*
330   *function_definition=: id(name) '(' (id(param_name) (',' id(param_name))*)? ')' '{' (statement)* '}'*
331

332   *statement =: expression ';'*
333   *| id(local_var) '=' expression ';'*
334   *| 'if' '(' expression ')' statement ('else' statement)?*
335   *| (id(label) ':')? 'while' '(' expression ')' statement*
336   *| (id(label) ':')? 'do' statement 'while' '(' expression ')' ';'*
337   *| (id(label) ':')? 'for' '(' initializer_list ';' expression ';' (expression (',' expression)*)? ')' statement*
338   *| '{' (statement)* '}'*
339   *| 'break;'*
340   *| 'return' expression ';'*
341   *exression =: ternary*
342   *ternary =: concatenation ('?' ternary ':' ternary)?*
343   *concatenation = logical ('_' logical)**
344   *logical =: comparison (('&&' | '||') comparison)**
345   *comparison =: math (('<' | '<=' | '>' | '>=' | '==' | '!=') math)?*
346   *math =: muls (('+' | '-' muls)**
347   *muls =: unary (('*' | '/' | '%' | '^' | '&' | '|' | '<<' | '>>') unary)**
348   *unary =: unary_head unary_tail*
349   *unary_head=: NUMERIC_CONST | STRING_LITERAL*
350   *| id(var_name)*
351   *| ('-' | '~' | '!' | '++' | '--') unary_head*
352   *| '(' expression ')'*
353   *| '[' id(field_name) (',' id(field_name))* ']' (('*' muls) / ('{' expression (',' expression)* '}'))?*
354   *'*
355   *unary_tail=: '[' expression (':' expression)?)']'*
356   *| (':=' | '+=' | '-=' | '/=' | '_=' ....) expression*
357   *| '(' expression (',' expression)* ')'*
358   *| '.' id(field_name)*
359   *| '++' | '--'*

360   The source codes of the programs are encoded in utf8.
361   The modeling language is a typed language. Types are not converted to each other
362 automatically. Each expression has a type known at the compilation stage. Types have no
363 names. The same types are those that have the same constructors and the same sets of
364 parameters.
365   The language supports the following data types: int (32 bits with a sign), bool (false | true),
366 string (characters in utf-8, addressed byte by byte), table (an array of variable size, containing
367 structures with fields), link to a function, array of variable size.
368   Variables in the language are set in the form: "name = initializer". The type of initializer
369 simultaneously sets the type of the variable. It is forbidden to declare an uninitialized variable.
370 Variables starting with "_" are reserved for platform variable names (for example, _currentTime).
371 A variable that has not been modified in the program code are considered as constant when
372 compiled.
373   Global variables are initialized before the function that starts the program is called. They are
374 initialized in the order of their declaration in the program. It is possible to access global variables

375 before declaring them. If during the initialization of global variables functions are called, then in
376 these functions, global variables that are not yet initialized cannot be used.

377   Changes in variables (assignment) is performed by a group of operators + =, & = b, etc. To
378 assign a new value to a variable, the operator:: = is used.

379   Local variables are defined in context from the declaration point to the end of the block in
380 which they are declared. Local variables declared in the header of the for loop are defined in
381 this header and the loop body.

382   Declaring two identical variables in the same context is not allowed. Also, overriding (hiding)
383 an external context variable is not allowed.

384   The language provides the following control structures: block of operators, declaration of a
385 local variable, expression, condition, loop, exit from the loop, exit from the function, and return
386 value.

387   The functions in the proposed modeling language are described in the form: 'name
388 (parameters) {actions}'. Nested local functions are not supported, as this leads to the need to
389 work with closures and increases the complexity of the program.

390   The result of the return expression determines the type of result. The types of parameters
391 are determined by the actual parameters passed in the first function call. The types of
392 parameters in all calls must match because using polymorphism or instantiating templates can
393 lead to an increase in the size of the generated code and to increase the time of program
394 execution.

395   Functions can be called before they are declared. Functions can be called directly, assigned
396 to variables, passed as parameters, and returned as results.

397   When working with strings, the following operations can be performed: assigning a literal
398 constant, concatenation, obtaining the length of a string, converting a string to a number,
399 converting a number to a string, receiving a substring, comparing strings, parsing a string and
400 patterns, searching for a string in a table, parsing parameters.

401   The following options are provided for working with the tables: create a table, create a table
402 with initialization, access fields, get the number of records, insert records, delete records,
403 search in the table by a numeric key, search in the table by a string key.

404   When working with arrays, it is possible to create them, create with initialization, access
405 elements, get the size, insert elements, delete elements, search for an element.

406   The data is encoded by the int data type and corresponds to Unix time.

407   Further extension of the developed language is possible, in particular, support of program
408 interfaces, class structures, inheritance mechanisms.

409
410 **Program models for monitoring.**

411 The program is translated into a binary format. The translation is provided through a developed
412 compiler. As a result of the translation, a program module is formed. During the conversion, the
413 compiler checks the syntax of the program text, the consistency of the names, and the
414 compatibility of the type.

415   When describing a program in binary format, instructions in the general command language
416 are used, which makes it possible to extend the developed language. Instructions are strictly
417 typed that allows high-speed program execution. The compactness of the binary programs is
418 ensured by using language structures that are close to the solved monitoring tasks and by the
compactness of the structures themselves.

419
420   Monitoring programs are intended for execution by the virtual machines. Loading and
421 unloading modules, their registration and calls assumed to be carried out through the software
interface, which is provided by the machines:

422   ```
int   vm_load_module (void* data); // data should be allocated with vm_alloc
```
423    ```
void   vm_free_module (int id);
```
424   ```
size_t vm_lookup   (char* fn_name);
```
425    ```
void   vm_call      (size_t ord, char* params);
```
426   ```
void *vm_alloc   (size_t size);
```
427
428    Modules cannot call each other's functions or exchange information.
429    When the module is loaded, the start () function that registers the named event handlers is
executed.
430    The predefined handlers are the following:
431    "t" – a timer handler that is called once a second in the idle mode;
432    "c" – a handler that is called in response to a console command;
433    "s' – a handler that is called, before sending any packet, allows adding the necessary fields
434 and events to the packet.
435    Parameters are passed to the handlers as text strings with separators "|". In the language,
436 the operators that allow extract parameters from the strings of this format are supported.

437    ```
// if called myHandler("123|Hello");
```
438   ```
myHandler(p)
```
439    ```
{
```
440   ```
  screen = intArg(); // screen = 123
```
441    ```
  mode = strArg();   // mode = "Hello";
```
442   ```
  ...
```
443    ```
}
```
444   The loadable modules have access to the sets of system variables, parameters provided by
445 system functions. They can initiate actions that can be performed by the system functions.

446   ```
TimerProc(s) {
```
447    ```
 if (_systemMemFree < 1024*1024)
```
448   ```
  reboot();
```
449    ```
}
```
450   When executing code in a virtual machine, the following verification procedures are
451 performed: availability of free memory, stack overflow error, looping, index out of array bounds
452 error, accessing uninitialized fields, division by zero, getting the remainder of division by zero.
453 When an exception occurs, the module is unloaded, and its resources are freed, then its handler
454 stops perceiving information about external events.

455 **A virtual machine for monitoring.**
456 The synthesized monitoring programs are executed by the virtual machine (VM) [29], whose
457 structure is shown in Fig. 2.
458   The main elements of the virtual machine are the following:
459   CN (Computer Network) Agent - an application running on a network element and
460 containing a virtual machine (CN VM) in which program modules are executed;
461   CN VM (Virtual Machine) – the environment for the program modules execution;
462   CM VM prodModule - an agent which logic is determined by the synthesized programs that
463 are loaded on the elements of the networks;
464   Logger – an element of the network that is responsible for collecting and providing log files
465 produced by the elements of the networks.

466
467 **Figure 2: Monitoring virtual machine structure.**

468 The logic of using the virtual machine is as follows. Program modules are compiled for
469 each of the synthesized programs. The modules are placed on the Data Provider. When new
470 modules appear, they are loaded into the VM. To load the modules, DTS is used. VM ensures
471 the execution of the modules. The logic of data gathering and data processing that is defined by
472 the loaded program modules can be executed once, in the loop, or on the event. The gathered
473 data is recorded during the execution of the module by the Logger. The collected data is
474 transmitted to the monitoring system. There is a possibility for messages exchanging between
475 the VM and the monitoring system. The CN Listener provides it. The monitoring system can
manage the virtual machine using the Command Manager.

476 ## V. Discussion

477 The proposed method for program synthesis can be used for a number of real-life applications.
478 The cable TV network monitoring system is taken as a typical example of a telecommunication
479 network. A common cable TV network contains one or more servers that serve TV receivers of
480 the end-users. The receivers provide users a variety of services for watching TV. The task of the
481 monitoring systems is to identify the errors that occur on the devices of the end-users, to
482 localize them, and find their causes.

483 Monitoring systems are deployed on the servers, and the virtual machines are installed on
484 TV receivers. Program modules for monitoring are synthesized on the servers and are executed
485 by the virtual machines.

486 The software of the TV receivers has a multi-level structure. The software of the upper
487 levels provides the logic of the user's services. On the low levels, the interaction with technical
488 means is implemented. The number and composition of the intermediate levels depend on the
489 type of receivers. The intermediate levels can include a level of the primary functions and a
490 transport level.

491 The state of each user service is characterized by several parameters related to different
492 levels of the software structure. There are links between the parameters of different levels.

493 The primary user services include image display services (Video), services of individual
494 delivery of television programs and films to subscribers (VOD (Video on Demand, VOD),
495 services for viewing private live broadcasts (PPV (Pay per View, PPV) and others. To the
496 essential services refer Electronic Program Guide service (EPG), service for writing data to disk
497 (DVR), authorization service (Authorization), service for switching between channels
498 (ChanelSW), etc. Transport level services allow receive/transmit data to external systems.
499 These services receive data from the tuner (Tuning service), decode incoming data streams
500 (Decoding service).

501 An error in the operation of any service results in malfunctioning of one or several end-user
502 services.

503 Regular monitoring assumes control of the parameters that characterize the state of the
504 end-user services, including Video, VOD, and PPV services. These parameters are NO_Video,
505 NO_VOD, NO_PPV. They take the value 'true' when errors occur in the working of these
506 services. If one of the parameters takes the value 'true', a message is formed on the TV receiver
507 and transmitted to the monitoring system. After receiving one such message, the monitoring
508 system synthesizes a new monitoring program for collecting data necessary to localize the error

509   and identify its cause. The synthesized program is compiled and loaded on the receiver that has
510   sent the message.
511   A model of the process of regular monitoring of a user service is presented in Fig. 3. The
512   monitoring process may be in one of the following states: initial state S0 and final state S1,
513   processing of a parameter S2, service is fully functional S3, service is malfunctioning S4,
514   sending an error message about service failure S5. The values of the controlled parameters
515   determine transitions between the states.
516   **Figure 3: Model of the regular monitoring process.**
517   If an error occurs, the newly synthesized programs provide a collection of additional
518   parameters related to services provided by the software of the lower levels. In this case, the
519   process model has a structure that is shown in Fig. 4. The following states are defined in the
520   model: initial state S0, final state S1, collection of an extended set of parameters S2, processing
521   of an extended set of parameters S3, service is fully functional S4, service is malfunctioning S5,
522   sending an error message about the service failure S6, sending collected additional parameters
523   S7.
524   **Figure 4: Model of the monitoring process in case an error occurs in the work of service.**
525   Consider the situation when there is no image on the user's screen, i.e., NO_Video error
526   has occurred. Other upper-level services, in particular, VOD and PPV, are operating normally.
527   Thus, the initial state of the users' device is described by the following vector of the parameters
528   {NO_Video, VOD, PPV}, the output vector is defined as {Video}. The output vector corresponds
529   to the state of the device when there is an image on the screen. Fig. 5 shows the scheme of the
530   synthesis that allows reaching the state {Video} from the state {NO_Video, VOD, PPV}. The
531   synthesis is carried out in two directions - direct and reverse. Within the direct synthesis, the
532   possibilities of transition from {NO_Video, VOD, PPV} to {Video} are considered. If such
533   possibilities exist, then reverse synthesis allows building programs of monitoring on the base of
534   identified possibilities.
535   Figure 5 shows the services of various levels and links between them. The descriptions of
536   the software structure determine the links between the services. The solid line indicates known
537   links that exist. The dotted lines show the links that should exist when the Video service is fully
538   functional.
539   According to the Fig.5, the result of the direct synthesis is the vector {DVR, EPG, Decoding,
540   Authorization, ChanelSW}. The state of the Tuning service is unknown, but it is required to
541   identify the possibility of reaching the state {Video}. Using the reverse synthesis, the sequence
542   {Video} -> {Tuning} -> {No_Video} is constructed. Following this sequence, it is necessary to
543   check the tuner's function of data receiving.
544   **Figure 5: The scheme of the synthesis of the monitoring programs to identify the cause**
545   **of no image on users' TV screens.**
546   A dozen parameters characterize the status of the tuner function of data received. The
547   types of tuners determine them. The synthesized monitoring program contains commands for
548   collecting the required set of parameters.
549   Below fragments of synthesized programs are presented. The following program is used for
550   regular monitoring of the state of the Video service:
551   *module main;*
552   *// event code assignment*
553   *NO_VIDEO_EVENT   = <event_code>;*

```
554    start(){
555    // event registration
556        register("dc:NoVideo()",NoVideo);
557    }
558
559    SendAlertNoVideo(){
560      alertTime = _currentTime;
561      {
562         reportInt(NO_VIDEO_EVENT);
563         reportInt(alertTime);
564         return true;
565      }
566      return false;
567    }
```

568   The program below provides the collection of additional parameters that allow find the
569 causes of no image on the users' screens:

```
570    module main;
571    start(){
572    // standard initialization procedures
573    ...
574    //gathering required parameters
575        reportInt(vm_ia_firstTunerParam);
576        reportInt(vm_ia_secondTunerParam);
577                            ...
578        reportInt(vm_ia_NTunerParam);
579    // standard termination procedures
580                            ...
581      }
```

582 VI. Conclusion

583 In the paper, the multilevel synthesis of programs for telecommunication networks monitoring is
584 discussed. A new method for monitoring dynamic networks with a multilevel structure is
585 proposed.

586   A new language for describing monitoring programs has been developed. The programs are
587 translated into binary code using a compiler. The compiled programs can be executed in
588 monitoring systems and on end-user devices. For this, a new virtual machine that is installed on
589 the elements of the network has been designed and implemented.

590   The value of the proposed multilevel synthesis is that it offers the possibility to build
591 reconfigurable monitoring systems for telecommunication networks.

592   The proposed method can be used for telecommunication networks monitoring in a wide
593 range of subject domains.

594   Further development of the ideas of multilevel synthesis of monitoring programs assumes
595 its application for a wide range of subject domains.

596

597 References

598 **Al Mamun MA, Hannan M, Hussain A, and Basri H. 2016.** Theoretical model and
599   implementation of a real time intelligent bin status monitoring system using rule based
600   decision algorithms. *Expert Systems with Applications* **48**:76-88.
601   10.1016/j.eswa.2015.11.025

602 **Baig MM, GholamHosseini H, Moqeem AA, Mirza F, and Lindén M. 2017.** A systematic
603   review of wearable patient monitoring systems–current challenges and opportunities for
604   clinical adoption. *Journal of medical systems* 41:115. DOI 10.1007/s10916-017-0760-1
605 **Bashar A. 2015.** Graphical modelling approach for monitoring and management of
606   telecommunication networks. *International Journal of Space-Based and Situated*
607   *Computing* 5:65-75. DOI 10.1504/ijssc.2015.069199
608 **Bello JP, Silva C, Nov O, DuBois RL, Arora A, Salamon J, Mydlarz C, and Doraiswamy H.**
609   **2018.** SONYC: A system for the monitoring, analysis and mitigation of urban noise
610   pollution. *arXiv preprint arXiv:180500889*.
611 **Cabal-Yepez E, Garcia-Ramirez AG, Romero-Troncoso RJ, Garcia-Perez A, and Osornio-**
612   **Rios RA. 2012.** Reconfigurable monitoring system for time-frequency analysis on
613   industrial equipment through STFT and DWT. *IEEE Transactions on Industrial*
614   *Informatics* 9:760-771. DOI 10.1109/tii.2012.2221131
615 **Chakravarthi MK, Tiwari RK, and Handa S. 2015.** Accelerometer based static gesture
616   recognition and mobile monitoring system using neural networks. *Procedia Computer*
617   *Science* 70:683-687. DOI 10.1016/j.procs.2015.10.105
618 **Cheng W-L, Chuang T-C, Yang C-W, Lin Y-H, Liu M, and Yin C. 2017.** An integrated security
619   monitoring system for digital service network devices. 2017 19th Asia-Pacific Network
620   Operations and Management Symposium (APNOMS): IEEE. p 118-122.
621 **Cong J, Liu B, Neuendorffer S, Noguera J, Vissers K, and Zhang Z. 2011.** High-level
622   synthesis for FPGAs: From prototyping to deployment. *IEEE Transactions on Computer-*
623   *Aided Design of Integrated Circuits and Systems* 30:473-491. DOI
624   10.1109/tcad.2011.2110592
625 **Cong J, and Minkovich K. 2007.** Optimality study of logic synthesis for LUT-based FPGAs.
626   *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*
627   **26**:230-239. DOI 10.1109/tcad.2006.887922
628 **Dovžan D, Logar V, and Škrjanc I. 2014.** Implementation of an evolving fuzzy model (eFuMo)
629   in a monitoring system for a waste-water treatment process. *IEEE Transactions on*
630   *Fuzzy Systems* 23:1761-1776. DOI 10.1109/tfuzz.2014.2379252
631 **Eissa M, Elmesalawy MM, and Hadhoud MM. 2015.** Wide area monitoring system based on
632   the third generation universal mobile telecommunication system (UMTS) for event
633   identification. *International Journal of Electrical Power & Energy Systems* 69:34-47. DOI
634   10.1016/j.ijepes.2014.12.077
635 **Fu H. 2012.** Equalization for high-speed serial interfaces in Xilinx 7 series FPGA transceivers.
636   *Xilinx white paper* 419.
637 **Gaikwad N, and Mistry Y. 2015.** Review on Environment Monitoring System and Energy
638   Efficiency. *International Journal of Engineering Research and Applications* 5:90-92.
639 **Gavrilov AV, Bezukladnikov II, and Kavalerov MV. 2017.** The problem of management and
640   monitoring in distributed telecommunication systems. 2017 IEEE Conference of Russian
641   Young Researchers in Electrical and Electronic Engineering (EIConRus): IEEE. p 140-
642   143.
643 **Ge Z, Song Z, and Gao F. 2013.** Review of recent research on data-based process monitoring.
644   *Industrial & Engineering Chemistry Research* 52:3543-3562. DOI 10.1021/ie302069q
645 **Koetter F, and Kochanowski M. 2015.** A model-driven approach for event-based business
646   process monitoring. *Information Systems and e-Business Management* 13:5-36. DOI
647   10.1007/s10257-014-0233-8
648 **Korableva O, Kalimullina O, and Kurbanova E. 2017.** Building the Monitoring Systems for
649   Complex Distributed Systems: Problems and Solutions. ICEIS (2). p 221-228.
650 **Kosinski J, Nawrocki P, Radziszowski D, Zielinski K, Zielinski S, Przybylski G, and Wnek**
651   **P. 2008.** SLA monitoring and management framework for telecommunication services.

652   Fourth International Conference on Networking and Services (icns 2008): IEEE. p 170-
653   175.
654 **Kulkarni P, McClean S, Parr G, and Black M. 2006.** Deploying MIB data mining for proactive
655   network management. 2006 3rd International IEEE Conference Intelligent Systems:
656   IEEE. p 506-511.
657 **Kumar A, Kim H, and Hancke GP. 2012.** Environmental monitoring systems: A review. *IEEE*
658   *sensors journal* **13**:1329-1339.
659 **Lyke JC, Christodoulou CG, Vera GA, and Edwards AH. 2015.** An introduction to
660   reconfigurable systems. *Proceedings of the IEEE* **103**:291-317.
661 **Lysecky R, Vahid F, and Tan SX-D. 2004.** Dynamic FPGA routing for just-in-time FPGA
662   compilation. Proceedings of the 41st annual Design Automation Conference: ACM. p
663   954-959.
664 **Maksymyuk T, Dumych S, Brych M, Satria D, and Jo M. 2017.** An IoT based monitoring
665   framework for software defined 5G mobile networks. IMCOM. p 105.
666 **Morgan J, and O'Donell G. 2014.** A service-oriented reconfigurable process monitoring
667   system-enabling cyber physical systems. *Journal of Machine Engineering* **14**. DOI
668 **Morgan J, and O'Donnell GE. 2018.** Cyber physical process monitoring systems. *Journal of*
669   *Intelligent Manufacturing* **29**:1317-1328.
670 **Mukhopadhyay SC. 2014.** Wearable sensors for human activity monitoring: A review. *IEEE*
671   *sensors journal* **15**:1321-1330. DOI 10.1109/jsen.2014.2370945
672 **Myint CZ, Gopal L, and Aung YL. 2017.** Reconfigurable smart water quality monitoring system
673   in IoT environment. 2017 IEEE/ACIS 16th International Conference on Computer and
674   Information Science (ICIS): IEEE. p 435-440.
675 **Osipov, and Yu. V. 2016.** Automatic synthesis of action programs for intelligent robots.
676   *Programming & Computer Software* **42**:155-160. DOI 10.1134/s0361768816030063
677 **Osipov VY, Vodyaho AI, Zhukova NA, and Glebovsky PA. 2017.** Multilevel automatic
678   synthesis of behavioral programs for smart devices. 2017 International Conference on
679   Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO): IEEE. p 335-340.
680 **Qi J, Wu F, Li L, and Shu H. 2007.** Artificial intelligence applications in the telecommunications
681   industry. *Expert Systems* **24**:271-291.
682 **Raju KRSR, and Varma GHK. 2017.** Knowledge based real time monitoring system for
683   aquaculture using IoT. 2017 IEEE 7th International Advance Computing Conference
684   (IACC): IEEE. p 318-321.
685 **Rubin R, and DeHon A. 2011.** Choose-your-own-adventure routing: lightweight load-time
686   defect avoidance. *ACM Transactions on Reconfigurable Technology and Systems*
687   *(TRETS)* **4**:33. DOI 10.1145/2068716.2068719
688 **Sakhardande P, Hanagal S, and Kulkarni S. 2016.** Design of disaster management system
689   using IoT based interconnected network with smart city monitoring. 2016 International
690   Conference on Internet of Things and Applications (IOTA): IEEE. p 185-190.
691 **Silva N, Marques ER, and Lopes L. 2018.** F lux: A Platform for Dynamically Reconfigurable
692   Mobile Crowd-Sensing. *ACM Transactions on Sensor Networks (TOSN)* **14**:20. DOI
693   10.1145/3200202
694 **Singh R, Gernaey KV, and Gani R. 2010.** An ontological knowledge-based system for the
695   selection of process monitoring and analysis tools. *Computers & chemical engineering*
696   **34**:1137-1154. DOI 10.1016/j.compchemeng.2010.04.011
697 **Sohn H, Farrar CR, Hemez FM, Shunk DD, Stinemates DW, Nadler BR, and Czarnecki JJ.**
698   **2003.** A review of structural health monitoring literature: 1996–2001. *Los Alamos*
699   *National Laboratory, USA*.
700 **Suma N, Samson SR, Saranya S, Shanmugapriya G, and Subhashri R. 2017.** IOT based
701   smart agriculture monitoring system. *International Journal on Recent and Innovation*
702   *Trends in computing and communication* **5**:177-181.

703 **Shichkina Y., Kataeva G., Irishina Y., Stanevich E. 2019.** The architecture of the system for
704   monitoring the status in patients with Parkinson's disease using mobile technologies, ,
705   *Studies in Computational Intelligence* **868**: 531-540
706 **Shichkina Y., Koblov A. 2019** The Use of Context-Dependent Modelling for the Construction of
707   an Anti-fraud System in Transport, *Lecture Notes in Computer Science (including*
708   *subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*
709   **11660:** 147-156
710 **Thompson A. 2012.** *Hardware Evolution: Automatic design of electronic circuits in*
711   *reconfigurable hardware by artificial evolution*: Springer Science & Business Media.
712 **Yang Z, Zhou Q, Lei L, Zheng K, and Xiang W. 2016.** An IoT-cloud based wearable ECG
713   monitoring system for smart healthcare. *Journal of medical systems* **40**:286. DOI
714   10.1007/s10916-016-0644-9
715 **Stefanov, K., Voevodin, V., Zhumatiy, S., & Voevodin, V. 2015.** Dynamically reconfigurable
716   distributed modular monitoring system for supercomputers (DiMMon). Procedia
717   Computer Science, 66, 625-634.
718

Figure 1

Figure 1. The structure of the monitoring process
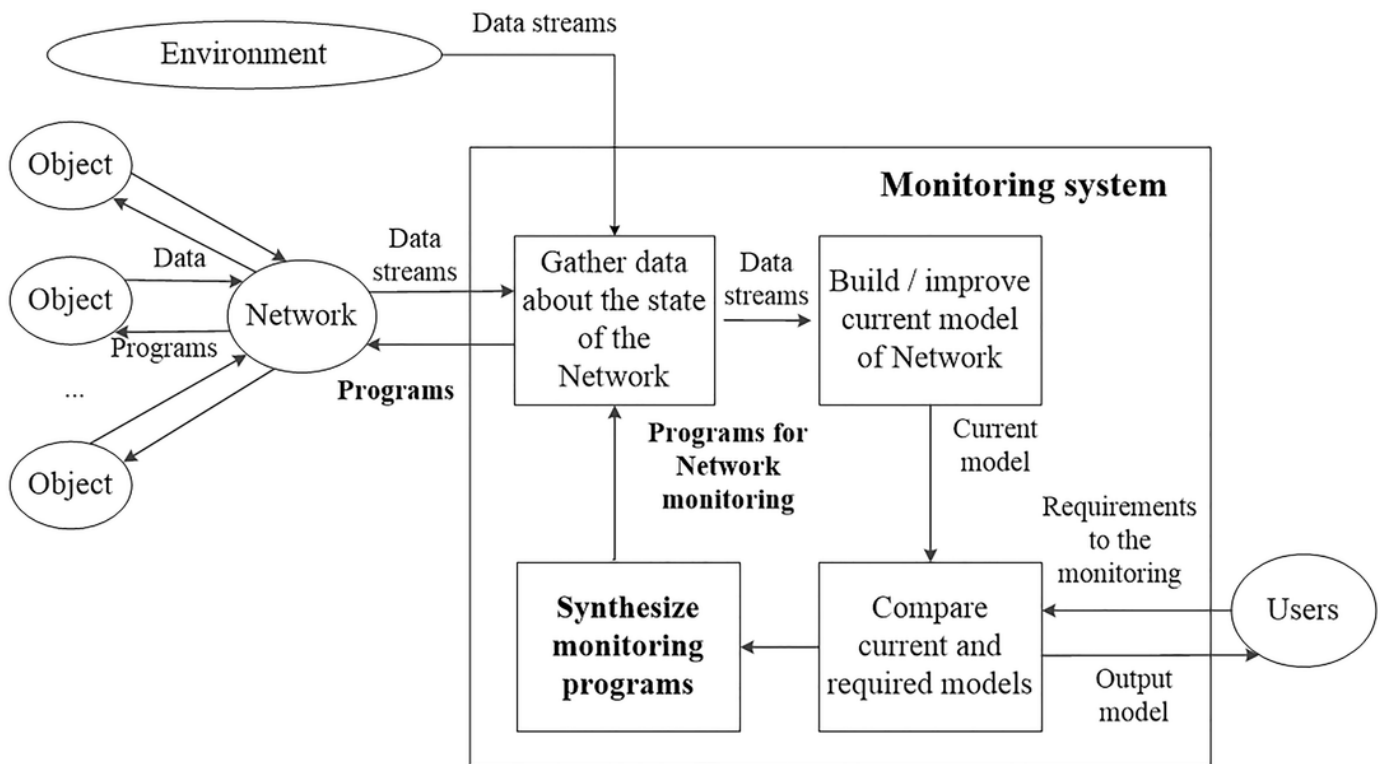
Figure 2

Figure 2. Monitoring virtual machine structure
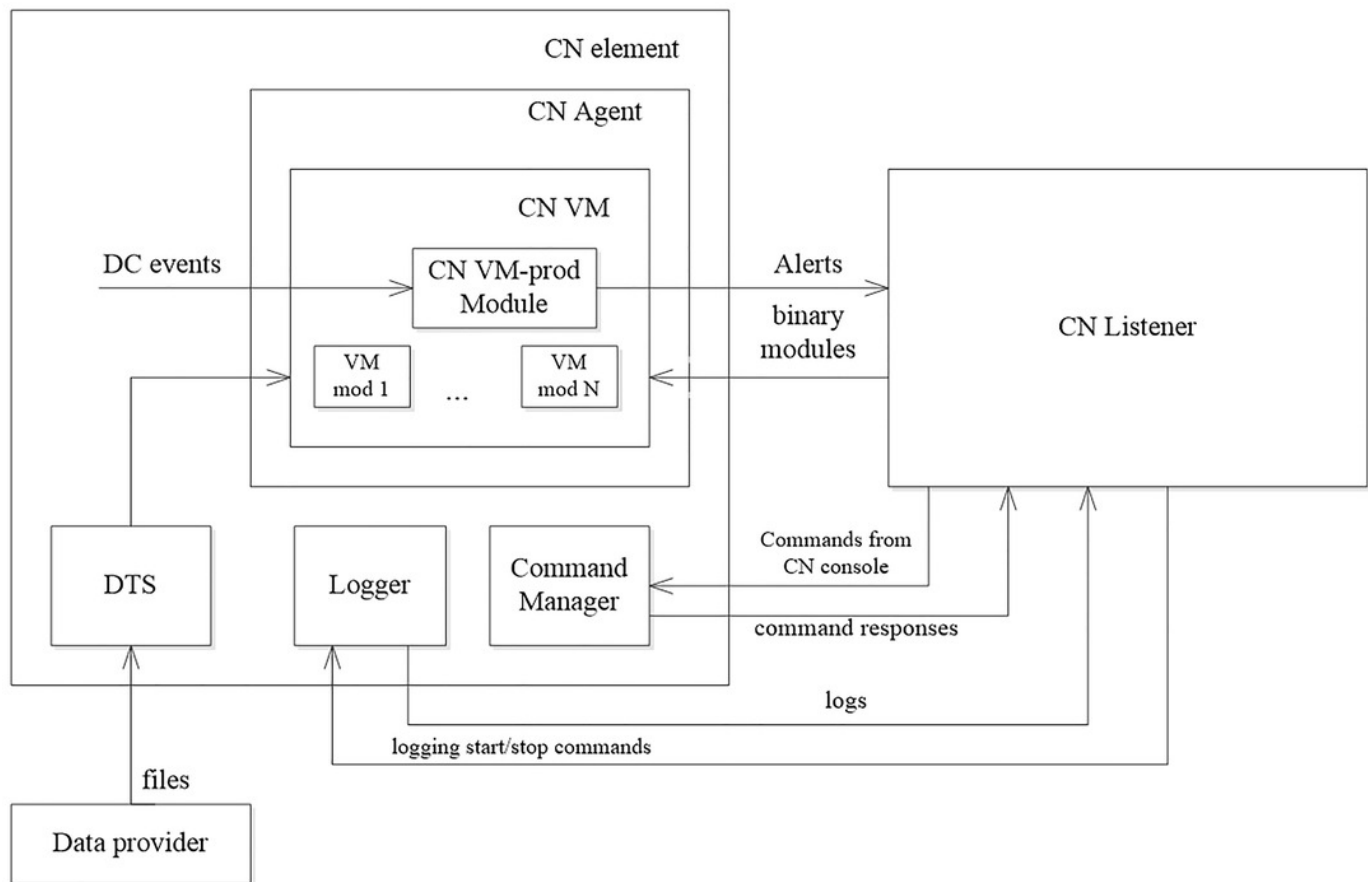
Figure 3

Figure 3. Model of the regular monitoring process
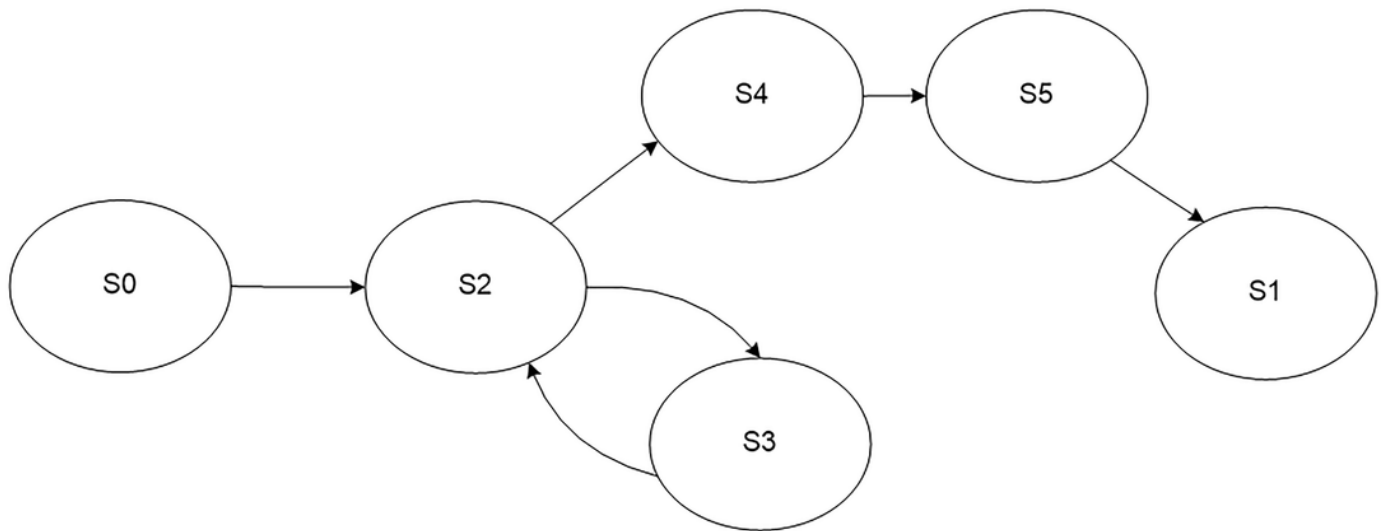
Figure 4

Figure 4. Model of the monitoring process in case an error occurs in the work of a service
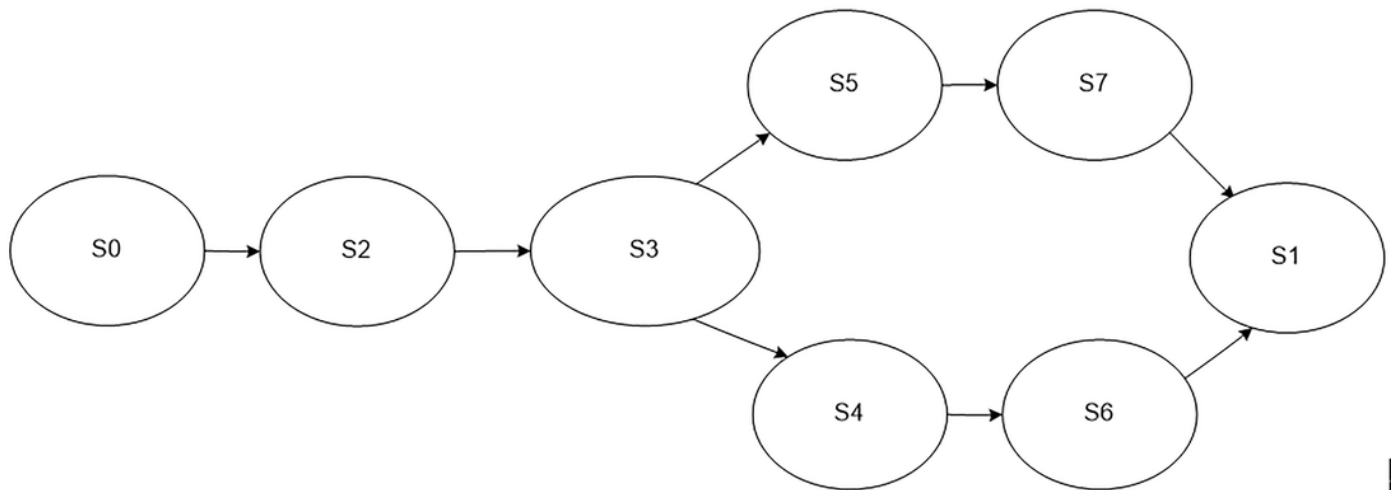
Figure 5

Figure 5. The scheme of the synthesis of the monitoring programs to identify the cause of no image on users TV screens
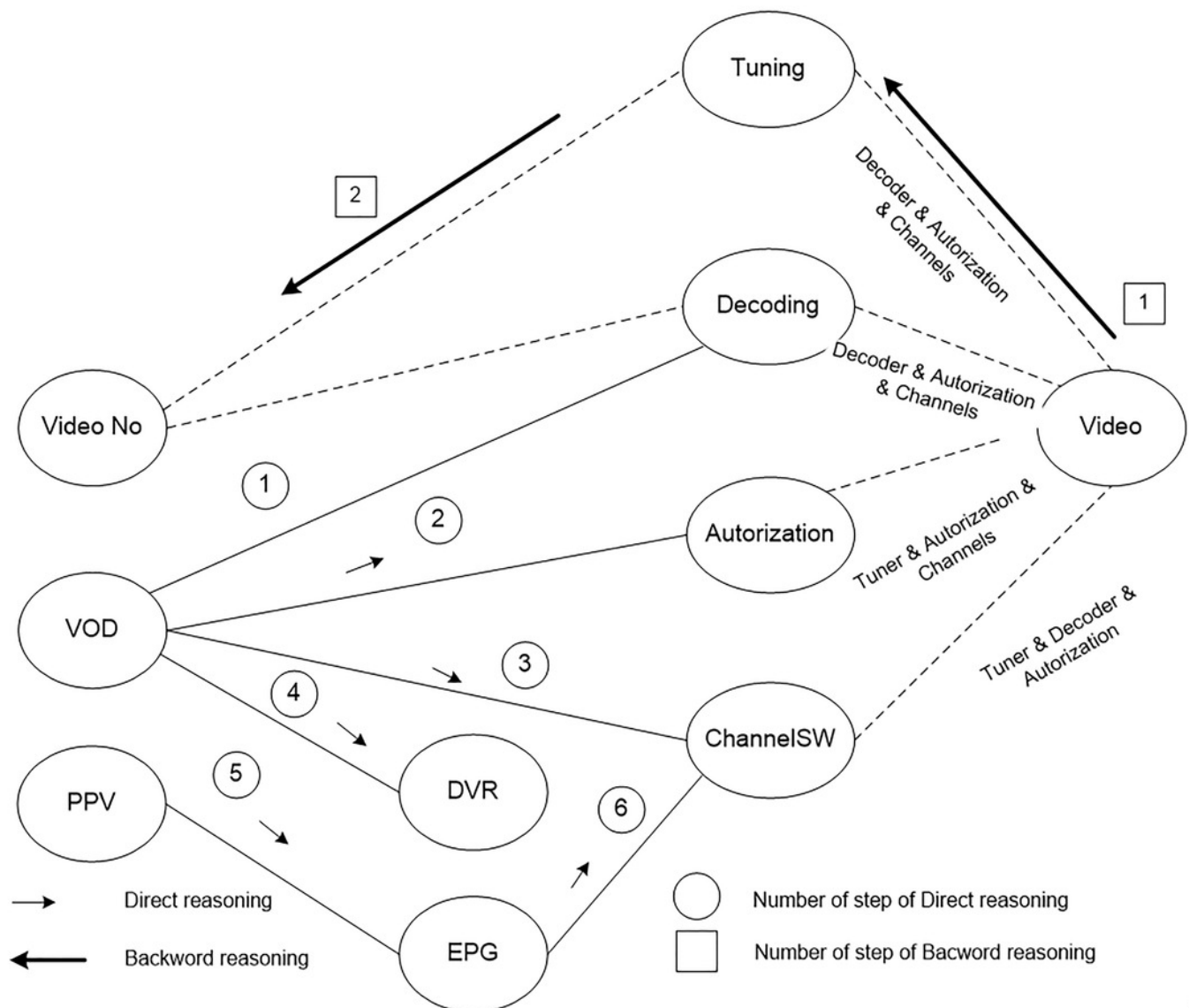
Table 1. Vocabulary of the modeling language

Symbol	Desription
space, tab, \n, \r	separator
// or /* */	comment
[A-Za-z_] [0-9A-Za-z_] *	identifier
\" \n \r \t \xNN \\ \0"	literal string
{}	block of code