Reconfigurable monitoring for telecommunication networks

Man Tianxing 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.
Reconfigurable monitoring for telecommunication networks

Man Tianxing¹, Vasily Yurievich Osipov², Alexander Ivanovich Vodyaho³, Andrey Kalmatskiy⁴, Natalia Alexandrovna Zhukova¹², Sergey Vyacheslavovich Lebedev³, Yulia Alexandrovna Shichkina³

¹ ITMO University, Saint Petersburg, Russia
² Saint-Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences, Saint Petersburg, Russia
³ St. Petersburg State Electrotechnical University, Saint Petersburg, Russia
⁴ Google, New York, America

Corresponding Author:
Man Tianxing¹
Kronverkskiy Prospekt, 49, St Petersburg, 197101, Russia
Email address: mantx626@gmail.com
Abstract
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 monitoring processes 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 presents 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.

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

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

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

For the reconfiguration of monitoring processes, business rules can be used. The rules define processes behavior for different known situations. This approach is well applicable for monitoring when networks have permanent structures and permanent conditions of use. The problem of monitoring of dynamic networks remains unsolved. To solve the problem monitoring systems must have abilities to build and rebuild the monitoring processes dynamically.
In the paper, we propose a new method for building monitoring processes based on using the theory of synthesis. To execute the processes a new language and a new virtual machine (VM) are developed.

The rest of the paper is structured as follows. In the second section, the capabilities of the existing monitoring systems are analyzed. It is shown that for dynamic telecommunication networks reconfigurable systems have not been considered till now. Then we describe monitoring processes and propose new models for monitoring programs and the method for their synthesis that allow develop reconfigurable monitoring systems. In the fourth section we present the implementation of the proposed method, which contains the description of the language for monitoring programs, a compiler for the language, and a virtual machine that is capable of executing the compiled monitoring programs on end-user devices. In the fifth section, a case study is given to prove the effectiveness of the method. The last section is the conclusion.

II. Materials & Methods

Related work

Today monitoring systems are used almost in all subject areas – medical (Baig et al. 2017; Mukhopadhyay 2014; Sohn et al. 2003; Yang et al. 2016, Shichkina et al. 2019), environment (Gaikwad & Mistry 2015; Kumar et al. 2012), social network (Chakravarthi et al. 2015), energy (Eissa et al. 2015), production (Singh et al. 2010), urban, economy (Bello et al. 2018; Sakhardande et al. 2016), agriculture (Suma et al. 2017), transport (Shichkina et al. 2019), fishery (Raju & Varma 2017), etc. For many subject areas, including telecommunication, monitoring systems are playing a critical role.

In traditional monitoring systems (Zabbix, http://www.zabbix.com/), (Nagios, http://www.nagios.org/), (Zenoss, https://www.zenoss.org/), (Pandora FMS https://pandorafms.com/), (Icinga https:// icinga.com/), (Sensu http://sensu.io/), (GroundWork Monitor https://www.gwos.com/), data is collected and then transmitted to monitoring systems in the form of the data streams. The data is processed by monitoring systems which have a distributed tree structure. Each element can collect and process data from multiple objects. The higher-level elements aggregate the data processing results of the lower-level elements. The tree structure allows the monitoring systems to have a sufficiently high level of performance. If the observed object changes and data is generated, redesign of the monitoring systems is a necessary process. Systems redesign assumes active involvement of software architects and programmers.

In the past few years, the monitoring systems have evolved significantly. Developed monitoring systems are based on the integration of multiple modern technologies (Morgan & O’Donnell 2018). In particular, in these systems, data is collected using Internet of Things technologies (Maksymyuk et al. 2017; Myint et al. 2017; Raju & Varma 2017; Sakhardande et al. 2016; Suma et al. 2017; Yang et al. 2016). Data processing employs statistical methods, data mining algorithms, and machine learning algorithms (Ge et al. 2013). These monitoring systems are a type of adaptive systems. They assume changes in both data gathering processes and processes of data processing. Adaptive systems use flexible data processing models (Cabal-Yepez et al. 2012; Dovžan et al. 2014) or business rules (Al Mamun et al. 2016; Koetter & Kochanowski 2015) to build and rebuild processes. The likelihood of their ability to
adapt in different cases depends on the composition of the selected business rules and the
defined values of customizable parameters of the processing models (Singh et al. 2010).
The reconfigurable monitoring systems have advanced capabilities for adaptive data
gathering and processing. The features of reconfigurable systems are discussed in (Korableva
et al. 2017; Lyke et al. 2015). In monitoring systems, reconfiguration operations can be
considered at several levels: sensor level, embedded system level, and application level. At the
sensor level, reconfiguration problems are solved by developing new programmable sensors
(Myint et al. 2017). The new programs are uploaded for updating the logic of the sensors. For
the adaptation of the embedded monitoring systems, they equip with reconfigurable data
preprocessing modules (Cabal-Yepez et al. 2012). At the application level, the adaptability is
achieved through customizable interfaces (Fu 2012), usage of genetic algorithms (Thompson
2012), and so on. It is also possible to synthesize new configurations for the systems and their
components at the level of field-programmable gate arrays (Cong et al. 2011; Cong & Minkovich
2007; Lysecky et al. 2004; Rubin & DeHon 2011).

For implementing reconfigurable systems, several methods have been proposed. In
(Morgan & O'Donell 2014), service-oriented architectures of reconfigurable systems are
presented. The problems of message routing and service orchestration in such systems are
discussed. Technological solutions for building reconfigurable systems are given in (Silva et al.
2018).

One of the limitations of the existing systems is that they cannot reconfigure themselves
dynamically. For dynamic reconfiguration, they have to be able to synthesize programs for data
gathering and data processing and execute them. Some of the processes should be performed
in the monitoring systems and some of the end-user devices.

To date, only a few dynamically reconfigurable systems exist. One of such systems has
been developed for monitoring the status and performance of a supercomputer network
(Stefanov, K., et al. 2015). The monitoring system is considered as a set of subsystems. The
subsystems process data about the performance of one or more tasks. Performance metrics are
calculated on the fly. To this purpose, a program module is dynamically created for each
process that is performed to solve the general task, and the transmission paths for the
calculated metrics are dynamically determined. The module oriented architecture of the systems
and their ability to create the modules dynamically make them able to adapt to new data
sources and new processing methods. When creating a new module, the nodes on which it will
be executed are selected based on the current load on the computer network. As a result, a
balanced load of the nodes is ensured. The nodes of the system can process a part of the data
about the state of the network. With these additional resources for solving the monitoring
problems are obtained. The results of using these monitoring systems have shown that they
allow solving the tasks of monitoring of supercomputer networks. Reconfigurable systems and
dynamically reconfigurable systems have not yet been developed in the field of
telecommunications.

III. Multilevel Synthesis of Monitoring Programs
Monitoring Objects, Processes, and Programs in Telecommunication Networks.
Telecommunication networks have a multilevel structure. The elements of the higher levels are
groups of objects. Within the lower levels, separate objects are considered. These objects are
end-user devices. The components and the subcomponents inside characterize them. There are multiple links between the elements at all levels.

Monitoring processes are oriented on providing data about the state of the end-users' devices to the customer service. These devices can gather data about their state and process it. Monitoring systems can interact with the devices. Within the interaction, data can be received from the devices, and commands on data gathering and data processing can be sent to them. The devices can perceive both separate commands and programs that contain multiple commands.

Monitoring programs are built and rebuild by monitoring systems. New programs are built in multiple cases, such as changes in the structure, state, or behavior of the devices or definitions of new requirements to the results of the monitoring. Requirements can refer to both the output data about the state of the network and the conditions of the monitoring, for example, admissible time modeling, accuracy and reliability of the models, consumed technical resources, etc.

There are two primary sources of data about the state of the devices. They are the messengers that are sent from the devices and the log files that they produce. The log files are sequences of records that describe events registered by the devices. Related data on factors affecting the network operation can be received from other sources.

The operators of the customer service define the requirements for the results of the monitoring. Figure 1 represents the structure of the monitoring process.

**Figure 1: The structure of the monitoring process.**

According to Fig 1, monitoring processes assume gathering data about the target objects (end-user devices). Current models of target objects are built using gathered data. These models reflect the state of the objects based on available data. Current models are compared with the required models. They are defined according to the requirements of the end-users (operators of customer service). Usually, these two models do not match. Current models can have many undefined parameters in comparison with the required models. If the models do not match, then new monitoring programs are synthesized. Possible ways of transitions between current and required models are considered to synthesize new programs. Each transition determines additional elements of data that are needed to reach the target model. Thus, after constructing sequences of transitions, it becomes known what data should be gathered. New monitoring programs contain sequences of commands that allow gathering required data elements.

The formally monitoring process $PR$ can be defined as $PR: O' \rightarrow O''$ where $O'$ is the model that reflects the state of the objects in the network. The model $O'$ is the current model that is build using gathered data. The model $O''$ is the model that is required by the user. The monitoring processes are targeted at building current models that match the required models.

The synthesis of monitoring programs is an essential part of the monitoring processes. Synthesized programs are defined as the complement of $O''$ concerning $O'$:

$$PRG = O'' \setminus O' = \{ e \in O'' | e \notin O' \} .$$

The synthesis of the monitoring programs is ensured through the following:

- The networks are commonly described in discrete time and discrete space. Thus automata are used as the models of the networks. Automata describe the states of the networks.
- Since there are too many possible states, some of them can be unknown, it is not possible
to describe all the states of the automata apriori. Due to that relatively finite automata are used (Osipov, V. Yu. et al. 2016). They are finite only for one moment of time. At each time step, all the parameters of the automata can be redefined.

- Traditional methods of synthesis have high computational complexity. The complexity can be reduced if we take into account the multilevel structure of the networks. To synthesize models of objects with multilevel structure multilevel methods of synthesis can be used.
- Compared to single-level methods, they allow increase the speed of the synthesis of the required programs in times. It is achieved due to the fact that the number of the elements in the models at upper levels is small in relation to the lower levels.
- The synthesized programs can be described using a new language. It is oriented on describing the programs for monitoring of telecommunication networks. The programs that are written using this language can be compiled and executed on end-user devices.

**Automata Models for Programs Synthesis.**

The models of the objects are multilevel structures that represent the objects as multiple sets of parameters. Sets of parameters and separate parameters are linked within separate levels and between the levels.

We suggest using multilevel automata models to synthesize the programs of monitoring for the objects that assume gathering information at different levels.

The model of the automata at one level can be defined in the following way (Osipov, V. Yu. et al. 2016; Osipov, V. Yu. et al. 2017):

\[ <AI, AO, AS, AR>, \]

where \( AI \) - a set of admissible input data, \( AO \) - a set of admissible output data, \( AS \) - a set of admissible internal states, \( AR \) - an admissible set of transition functions between the elements of the set of input data and the elements of the set of output data.

So, the entire multilevel model is

\[ < <AI^0, AO^0, AS^0, AR^0> ; <AI^1, AO^1, AS^1, AR^1> ; \ldots ; <AI^N, AO^N, AS^N, AR^N> > , \]

where the super index defines the level of the automata model.

At each step of program synthesis, automata can be in one of the admissible states. Within each step, the sets of admissible states and transition functions are rebuilt, taking into account the achieved state and the conditions of synthesis. Thus the automata at step \( r \) is defined as:

\[ <I_l, O_l, S_l, F_l, AI_{l-1}, AO_{l-1}, AS_{l-1}, AR_{l-1}>, \]

where \( I \) - input data, \( O \) - output data, \( S \) - internal state, \( F \) - transition functions (conditions). Parameters \( I, O, S, R \) are defined for the step \( r \) and parameters \( AI, AO, AS, AR \) for the step \( r-1 \).

**Method of Synthesis of Monitoring Programs.**

The method allows synthesizing programs for objects monitoring in conditions when the current and the target models of the objects are defined. It is required to gather data about the objects that are necessary to move towards required models from current models.

The method uses direct and backward inference. The direct inference is used to prove that the transition between models is possible and that the target model exists. As a result of the direct proof, multiple sequences of transitions can be defined. The backward inference allows restoring programs from the results of the proof. The inference is carried out according to the scheme mirrored to direct inference.

In the conditions, when the existence of the target models cannot be proved, the possibilities of changing the models can be considered.
In the automata model for program synthesis, the set of input data elements \( \{d_s\} \) are defined according to the current model of the object, the set of output data elements \( \{d_w\} \) are elements of the target model of the object. Transition functions \( F_{\text{zw}} \) reflect links between elements of the models: \( F_{\text{zw}}(d_{\text{zve}}) ->(d_{\text{zva}}) \), where \( e = [1, E_{\text{z}}] \) indicates elements of the models that are required to achieve an element indicated as \( d_{\text{zva}} \).

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

Below the description of the method is given:

**Input data.** \( \{d_s\} \) - the sets of the input data elements; \( \{d_w\} \) - the sets of the output data elements; \( F_{\text{zw}} \) - transition functions between the elements.

**Output data.** TransitionSequence - sequences of transitions that allow reach elements \( \{d_w\} \) of the target model having elements \( \{d_s\} \) of the current model.

The pseudo code of the method is the following:

```python
Function FindAchievableElements(maxLevel, M, CurrentM, TargetM) is
    for each l in [maxLevel..0] do
        for each i in [0..M.funConditions[l].length] do
            if checkCondition(M.funConditions[l][i]) == true do
                elementPairs = cartesianProduct(CurrentM.Elements[l], TargetM.Elements[l])
                for each j in [0..elementPairs.length]
                    if profTransition(M.vzLinkingFun[l][i](elementPairs[j].first, elementPairs[j].second) =true do
                        M.AchievedFrom[l] = add(M.AchievedFrom[l], elementPairs[j].first)
                        M.AchievedTo[l] = add(M.AchievedTo[l], elementPairs[j].first)
                        M.AchievedWith[l] = add(M.AchievedWith[l], M.vzLinkingFun[l][i])
                        M.AchievedWhile[l] = add(M.AchievedWhile[l], checkCondition[l][i])
                        remove(M.NonProven[l+1], elementPairs[j].second)
                    else
                        M.NonProven[l] = add(M.NonProven[l], elementPairs[j].second)
                    end
            end
        end
    if M.NonProven[l].length != 0 && maxLevel != 0 do
        FindAchievableElements(maxLevel-1, M, CurrentM, TargetM)
    end
    return M

Function DirectProgramSynthesis(maxLevel, M, CurrentM, TargetM) is
    M = FindAchievableElements(maxLevel, M, CurrentM, TargetM)
    if M.NonProven.length == 0 do
        for each l in [0..maxLevel] do
            M.TransitionSequence = zip(zip(M.AchievedFrom[l], M.AchievedWhile[l]), zip(M.AchievedTo[l], M.AchievedWith[l])) /* to get a list of quads of the form (function, condition, input, output)*/
        else error 'no transition is found'
    return M
```

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

Below the description of the method is given.

**Input data.** \( \{d_w\} \) - the sets of the input data elements; \( \{d_s\} \) - the sets of the output data elements. Target data elements are considered as the input, and the input data elements as the target elements because the inference is backward.

**Output data.** PRG -models of monitoring programs.

The pseudo-code of the method is the following:

```python
Function BackwdProgramSynthesis(maxLevel, M, CurrentM, TargetM) is
```

PeerJ Comput. Sci. reviewing PDF | (CS-2020:01:44774:2:0:NEW 4 Jun 2020)
PRG = initializeProgram
MProc = initializeProgramModel

/* Build the model of monitoring program using backward inference, TargetM and CurrentM are swapped*/
MProc = DirectProgramSynthesis(maxLevel, MProc, TargetM, CurrentM)
MProg = reproduceTransitionSequences(MProc)

/* Build the model of the monitoring program according to the synthesized structure */
for each l in [(maxLevel-1)..0] do
  for each i in [0.. MProc.AchievedWith[l].length] do
    z = getFunKind(MProc.AchievedWith[l][i])
    v = getFunType(MProc.AchievedWith[l][i])
    MProg.StructureElements[l] = add(MProg.StructureElements[l], buildStructureElement(z, v))
  end
  for each i in [0.. MProc.AchivedWhile[l].length] do
    conditions = MProc.AchivedWhile[l][i]
    MProg.Condition[l] = add(MProg.Condition[l], buildConditions(conditions))
  end
  MProg.Program[l] = buildProgramForLevel(MProg.StructureElements[l], MProg.Condition[l])
  PRG = buildProgram(PRG, MProg.Program[l])
end
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 $T_u$ required for program synthesis using the proposed method can be estimated by the following formula:

$$T_u \approx c \sum_{i=0}^{K} m_i^2 \leq c \sum_{i=0}^{K} m_i^2$$  \hspace{1cm} (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,

$$T_l \approx c \sum_{i=0}^{K} \frac{m_i^2}{n_i^2} \leq c \sum_{i=0}^{K} m_i^2$$  \hspace{1cm} (2)

The average estimate of time $T$ of multilevel synthesis concerning (1), (2) is equal to

$$T = (T_u + T_l) / 2.$$

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.
Table 1: Vocabulary of the modeling language.

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

```
program ::= 'module' id ';' (global_var_definition | function_definition)* <end of file>

global_var_definition ::= id(expression) '=' expression ';'

function_definition ::= id(param_name) '(' (id(param_name) (',' id(param_name))*)? ')' '{' (statement)* '}'

statement ::= expression ';

| id(local_var) '=' expression ';
| 'if' '(' expression ')' statement ('else' statement)?
| (id(label) ':')? 'while' '(' expression ')' statement
| (id(label) ':')? 'do' statement while '(' expression ')' '
| (id(label) ':')? 'for' '(' initialization_list ';''expression ';''(expression ('expression')? ')' statement
| '{' (statement)* '}'
| 'break';
| 'return' expression ';

expression ::= ternary

ternary ::= concatenation ('?''ternary''::*ternary)'?

concatenation = logical ("" logical)*

logical ::= comparison ("&&" | "||") comparison*

comparison ::= math ("<" | "<=" | ">" | ">=" | ">=" | "!=") math?

math ::= muls ("+" | "-" muls)*

muls ::= unary ("*" | "/" | ">" | ">=" | ">") muls)

unary ::= unary_head unary_tail

unary_head::= NUMERIC_CONST | STRING_LITERAL

| id(var_name)
| ("-" | "~" | "+" | "--") unary_head
| ("='" | "+='" | "+='") unary_head

unary_tail ::= unary_head

expression ("='" expression)? ')
| expression ("='" expression)*' 
| expression ("='" expression)*'

| 'id(field_name)
| '+=' | "+='. 

The source codes of the programs are encoded in utf8.

The modeling language is a typed language. Types are not converted to each other automatically. Each expression has a type known at the compilation stage. Types have no names. The same types are those that have the same constructors and the same sets of parameters.

The language supports the following data types: int (32 bits with a sign), bool (false | true), string (characters in utf-8, addressed byte by byte), table (an array of variable size, containing structures with fields), link to a function, array of variable size.

Variables in the language are set in the form: “name = initializer”. The type of initializer simultaneously sets the type of the variable. It is forbidden to declare an uninitialized variable.

Variables starting with "_" are reserved for platform variable names (for example, _currentTime).

A variable that has not been modified in the program code are considered as constant when compiled.

Global variables are initialized before the function that starts the program is called. They are initialized in the order of their declaration in the program. It is possible to access global variables...
before declaring them. If during the initialization of global variables functions are called, then in
these functions, global variables that are not yet initialized cannot be used.

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

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

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

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

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

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

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

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

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

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

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

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

**Program models for monitoring.**

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

When describing a program in binary format, instructions in the general command language
are used, which makes it possible to extend the developed language. Instructions are strictly
typed that allows high-speed program execution. The compactness of the binary programs is
ensured by using language structures that are close to the solved monitoring tasks and by the
compactness of the structures themselves.
Monitoring programs are intended for execution by the virtual machines. Loading and unloading modules, their registration and calls assumed to be carried out through the software interface, which is provided by the machines:

```c
int vm_load_module (void* data); // data should be allocated with vm_alloc
void vm_free_module (int id);
size_t vm_lookup (char* fn_name);
void vm_call (size_t ord, char* params);
void *vm_alloc (size_t size);
```

Modules cannot call each other's functions or exchange information. When the module is loaded, the start () function that registers the named event handlers is executed.

The predefined handlers are the following:
- "t" – a timer handler that is called once a second in the idle mode;
- "c" – a handler that is called in response to a console command;
- "s" – a handler that is called, before sending any packet, allows adding the necessary fields and events to the packet.

Parameters are passed to the handlers as text strings with separators "|". In the language, the operators that allow extract parameters from the strings of this format are supported.

```c
// if called myHandler("123|Hello");
myHandler(p)
{
    screen = intArg(); // screen = 123
    mode = strArg(); // mode = "Hello"
    ...
}
```

The loadable modules have access to the sets of system variables, parameters provided by system functions. They can initiate actions that can be performed by the system functions.

```c
TimerProc(s) {
    if (_systemMemFree < 1024*1024)
        reboot();
}
```

When executing code in a virtual machine, the following verification procedures are performed: availability of free memory, stack overflow error, looping, index out of array bounds error, accessing uninitialized fields, division by zero, getting the remainder of division by zero. When an exception occurs, the module is unloaded, and its resources are freed, then its handler stops perceiving information about external events.

**A virtual machine for monitoring.**

The synthesized monitoring programs are executed by the virtual machine (VM) [29], whose structure is shown in Fig. 2.

The main elements of the virtual machine are the following:
- CN (Computer Network) Agent - an application running on a network element and containing a virtual machine (CN VM) in which program modules are executed;
- CN VM (Virtual Machine) – the environment for the program modules execution;
- CM VM prodModule - an agent which logic is determined by the synthesized programs that are loaded on the elements of the networks;
- Logger – an element of the network that is responsible for collecting and providing log files produced by the elements of the networks.
Figure 2: Monitoring virtual machine structure.

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

V. Discussion

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

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

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

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

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

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

Regular monitoring assumes control of the parameters that characterize the state of the end-user services, including Video, VOD, and PPV services. These parameters are NO_Video, NO_VOD, NO_PPV. They take the value ‘true’ when errors occur in the working of these services. If one of the parameters takes the value ‘true’, a message is formed on the TV receiver and transmitted to the monitoring system. After receiving one such message, the monitoring system synthesizes a new monitoring program for collecting data necessary to localize the error.
and identify its cause. The synthesized program is compiled and loaded on the receiver that has
sent the message.

A model of the process of regular monitoring of a user service is presented in Fig. 3. The
monitoring process may be in one of the following states: initial state S0 and final state S1,
processing of a parameter S2, service is fully functional S3, service is malfunctioning S4,
sending an error message about service failure S5. The values of the controlled parameters
determine transitions between the states.

**Figure 3: Model of the regular monitoring process.**

If an error occurs, the newly synthesized programs provide a collection of additional
parameters related to services provided by the software of the lower levels. In this case, the
process model has a structure that is shown in Fig. 4. The following states are defined in the
model: initial state S0, final state S1, collection of an extended set of parameters S2, processing
of an extended set of parameters S3, service is fully functional S4, service is malfunctioning S5,
sending an error message about the service failure S6, sending collected additional parameters S7.

**Figure 4: Model of the monitoring process in case an error occurs in the work of service.**

Consider the situation when there is no image on the user's screen, i.e., NO_Video error
has occurred. Other upper-level services, in particular, VOD and PPV, are operating normally.
Thus, the initial state of the users' device is described by the following vector of the parameters
{NO_Video, VOD, PPV}, the output vector is defined as {Video}. The output vector corresponds
to the state of the device when there is an image on the screen. Fig. 5 shows the scheme of the
synthesis that allows reaching the state {Video} from the state {NO_Video, VOD, PPV}. The
synthesis is carried out in two directions - direct and reverse. Within the direct synthesis, the
possibilities of transition from {NO_Video, VOD, PPV} to {Video} are considered. If such
possibilities exist, then reverse synthesis allows building programs of monitoring on the base of
identified possibilities.

**Figure 5: The scheme of the synthesis of the monitoring programs to identify the cause
of no image on users' TV screens.**

According to the Fig.5, the result of the direct synthesis is the vector {DVR, EPG, Decoding,
Authorization, ChanelSW}. The state of the Tuning service is unknown, but it is required to
identify the possibility of reaching the state {Video}. Using the reverse synthesis, the sequence
{Video} -> {Tuning} -> {No_Video} is constructed. Following this sequence, it is necessary to
check the tuner's function of data receiving.

Below fragments of synthesized programs are presented. The following program is used for
regular monitoring of the state of the Video service:

```plaintext
module main;
// event code assignment
NO_VIDEO_EVENT = <event_code>;
```

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

```java
module main;

start(){
    // standard initialization procedures
    ...
    // gathering required parameters
    reportInt(vm_ia_firstTunerParam);
    reportInt(vm_ia_secondTunerParam);
    ...  
    reportInt(vm_ia_NTunerParam);
    // standard termination procedures
    ...
}
```

VI. Conclusion

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

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

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

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

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

References

Al Mamun MA, Hannan M, Hussain A, and Basri H. 2016. Theoretical model and implementation of a real time intelligent bin status monitoring system using rule based decision algorithms. Expert Systems with Applications 48:76-88.

10.1016/j.eswa.2015.11.025
Baig MM, GholamHosseini H, Moqeeem AA, Mirza F, and Lindén M. 2017. A systematic review of wearable patient monitoring systems—current challenges and opportunities for clinical adoption. *Journal of medical systems* 41:115. DOI 10.1007/s10916-017-0760-1

Bashar A. 2015. Graphical modelling approach for monitoring and management of telecommunication networks. *International Journal of Space-Based and Situated Computing* 5:65-75. DOI 10.1504/ijssc.2015.069199

Bello JP, Silva C, Nov O, DuBois RL, Arora A, Salamon J, Mydlarz C, and Doraiswamy H. 2018. SONYC: A system for the monitoring, analysis and mitigation of urban noise pollution. *arXiv preprint arXiv:180500889.*

Cabal-Yepez E, Garcia-Ramirez AG, Romero-Troncoso RJ, Garcia-Perez A, and Osornio-Rios RA. 2012. Reconfigurable monitoring system for time-frequency analysis on industrial equipment through STFT and DWT. *IEEE Transactions on Industrial Informatics* 9:760-771. DOI 10.1109/tii.2012.2221131

Chakravarthi MK, Tiwari RK, and Handa S. 2015. Accelerometer based static gesture recognition and mobile monitoring system using neural networks. *Procedia Computer Science* 70:683-687. DOI 10.1016/j.procs.2015.10.105

Cheng W-L, Chuang T-C, Yang C-W, Lin Y-H, Liu M, and Yin C. 2017. An integrated security monitoring system for digital service network devices. 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS): IEEE. p 118-122.

Cong J, Liu B, Neuendorffer S, Noguera J, Vissers K, and Zhang Z. 2011. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30:473-491. DOI 10.1109/tcad.2011.2110592

Cong J, and Minkovich K. 2007. Optimality study of logic synthesis for LUT-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26:230-239. DOI 10.1109/tcad.2006.887922

Dovžan D, Logar V, and Škrjanc I. 2014. Implementation of an evolving fuzzy model (eFuMo) in a monitoring system for a waste-water treatment process. *IEEE Transactions on Fuzzy Systems* 23:1761-1776. DOI 10.1109/ftfuzz.2014.2379252

Eissa M, Elmesalawy MM, and Hadhoud MM. 2015. Wide area monitoring system based on the third generation universal mobile telecommunication system (UMTS) for event identification. *International Journal of Electrical Power & Energy Systems* 69:34-47. DOI 10.1016/j.ijepes.2014.12.077

Fu H. 2012. Equalization for high-speed serial interfaces in Xilinx 7 series FPGA transceivers. *Xilinx white paper 419.*

Gaikwad N, and Mistry Y. 2015. Review on Environment Monitoring System and Energy Efficiency. *International Journal of Engineering Research and Applications* 5:90-92.

Gavrilov AV, Bezukladnikov II, and Kavalerov MV. 2017. The problem of management and monitoring in distributed telecommunication systems. 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus): IEEE. p 140-143.

Ge Z, Song Z, and Gao F. 2013. Review of recent research on data-based process monitoring. *Industrial & Engineering Chemistry Research* 52:3543-3562. DOI 10.1021/ie302069q

Koetter F, and Kochanowski M. 2015. A model-driven approach for event-based business process monitoring. *Information Systems and e-Business Management* 13:5-36. DOI 10.1007/s10257-014-0233-8

Korableva O, Kalimullina O, and Kurbanova E. 2017. Building the Monitoring Systems for Complex Distributed Systems: Problems and Solutions. ICEIS (2). p 221-228.

Kosinski J, Nawrocki P, Radziszowski D, Zielinski K, Zielinski S, Przybyski G, and Wnek P. 2008. SLA monitoring and management framework for telecommunication services.
Kulkarni P, McClean S, Parr G, and Black M. 2006. Deploying MIB data mining for proactive network management. 2006 3rd International IEEE Conference Intelligent Systems: IEEE. p 506-511.

Kumar A, Kim H, and Hancke GP. 2012. Environmental monitoring systems: A review. *IEEE sensors journal* 13:1329-1339.

Lyke JC, Christodoulou CG, Vera GA, and Edwards AH. 2015. An introduction to reconfigurable systems. *Proceedings of the IEEE* 103:291-317.

Lysecky R, Vahid F, and Tan SX-D. 2004. Dynamic FPGA routing for just-in-time FPGA compilation. Proceedings of the 41st annual Design Automation Conference: ACM. p 954-959.

Maksymyuk T, Dumych S, Brych M, Satria D, and Jo M. 2017. An IoT based monitoring framework for software defined 5G mobile networks. IMCOM. p 105.

Morgan J, and O'Donell GE. 2018. Cyber physical process monitoring systems. *Journal of Intelligent Manufacturing* 29:1317-1328.

Mukhopadhyay SC. 2014. Wearable sensors for human activity monitoring: A review. *IEEE sensors journal* 15:1321-1330. DOI 10.1109/jsen.2014.2370945

Myint CZ, Gopal L, and Aung YL. 2017. Reconfigurable smart water quality monitoring system in IoT environment. 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS): IEEE. p 435-440.

Osipov VY, Vodyaho AI, Zhukova NA, and Glebovsky PA. 2017. Multilevel automatic synthesis of behavioral programs for smart devices. 2017 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO): IEEE. p 335-340.

Qi J, Wu F, Li L, and Shu H. 2007. Artificial intelligence applications in the telecommunications industry. *Expert Systems* 24:271-291.

Raju KRSR, and Varma GHK. 2017. Knowledge based real time monitoring system for aquaculture using IoT. 2017 IEEE 7th International Advance Computing Conference (IACC): IEEE. p 318-321.

Rubin R, and DeHon A. 2011. Choose-your-own-adventure routing: lightweight load-time defect avoidance. *ACM Transactions on Reconfigurable Technology and Systems* (TRETS) 4:33. DOI 10.1145/2068716.2068719

Sakhardande P, Hanagal S, and Kulkarni S. 2016. Design of disaster management system using IoT based interconnected network with smart city monitoring. 2016 International Conference on Things and Applications (IOTA): IEEE. p 185-190.

Silva N, Marques ER, and Lopes L. 2018. F lux: A Platform for Dynamically Reconfigurable Mobile Crowd-Sensing. *ACM Transactions on Sensor Networks (TOSN)* 14:20. DOI 10.1145/3200202

Singh R, Gernaey KV, and Gani R. 2010. An ontological knowledge-based system for the selection of process monitoring and analysis tools. *Computers & chemical engineering* 34:1137-1154. DOI 10.1016/j.compchemeng.2010.04.011

Sohn H, Farrar CR, Hemez FM, Shunk DD, Stinemates DW, Nadler BR, and Czarnecki JJ. 2003. A review of structural health monitoring literature: 1996–2001. *Los Alamos National Laboratory*, USA.

Suma N, Samson SR, Saranya S, Shanmugapriya G, and Subhashri R. 2017. IOT based smart agriculture monitoring system. *International Journal on Recent and Innovation Trends in computing and communication* 5:177-181.
Shichkina Y., Kataeva G., Irishina Y., Stanevich E. 2019. The architecture of the system for monitoring the status in patients with Parkinson's disease using mobile technologies, *Studies in Computational Intelligence* 868: 531-540

Shichkina Y., Koblov A. 2019. The Use of Context-Dependent Modelling for the Construction of an Anti-fraud System in Transport, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11660: 147-156

Thompson A. 2012. *Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*: Springer Science & Business Media.

Yang Z, Zhou Q, Lei L, Zheng K, and Xiang W. 2016. An IoT-cloud based wearable ECG monitoring system for smart healthcare. *Journal of medical systems* 40:286. DOI 10.1007/s10916-016-0644-9

Stefanov, K., Voevodin, V., Zhumatiy, S., & Voevodin, V. 2015. Dynamically reconfigurable distributed modular monitoring system for supercomputers (DiMMon). *Procedia Computer Science*, 66, 625-634.
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. 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

(on next page)
| Symbol | Description |
|--------|-------------|
| space, tab, \n, \r | separator |
| // or /* */ | comment |
| [A-Za-z_] [0-9A-Za-z_] * | identifier |
| \\
| literal string |
| {} | block of code |