About one approach to multilevel behavioral program synthesis for television devices

Vasiliy Yu. Osipov, Natalia A. Zhukova, Alexander I. Vodyaho

***Abstract*—**An approach for multilevel synthesis of programs is suggested. The mathematical model of reconfigurable program in a form of relative finite state operational automata is introduced. On the base of this model the method of multilevel automatic synthesis is developed. Suggested approach can be used for generation of behavioral programs for set-top boxs in the domain of cable television.

***Keywords*—**Multilevel behavioral program synthesis, finite state operational automata, smart home devices

I. Introduction

Nowadays one of the main trends of IT industry development is increasing of the level of intelligence of the very wide class of devices, which are used in every day life such as devices for remote TV control, vacuum cleaner robot, etc. For building smart devices, as a rule, intelligent technologies, developed earlier for knowledge-intensive domains are used. For example modern space technologies assume usage of artificial intelligence on all stages of preparing of rockets for lunching, in the oil production industry artificial intelligence technologies are also widely used especially when oil fields are remote and hardly accessible. From the technical point of view, porting of technologies is not very sophisticated problem, but very often these solutions are too complex for home devices and the process of supporting of smart home devices is much more difficult, taking into account very hard requirement to total cost of ownership. Support assumes regular estimation of device status and device repairing after failures.

Vasiliy Yu. Osipov is with the Saint-Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS), 39, 14 Line, St. Petersburg, 199178 Russia, osipov_vasiliy@mail.ru

Natalia A. Zhukova is with the the Zodiac Interactive LLC, USA – Russia, 11, Sedova str., St. Petersburg, 192019, Russia, nazhukova@mail.ru, pietr@mail.ru

Alexander I. Vodyaho is with the St. Petersburg National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, Russia Zodiac, nazhukova@mail.ru

For complex unique systems such as oil processing systems special monitoring and emergency situation reaction, as a rule, unique algorithms are developed. But development of specialized algorithms for all types and all modifications of home devices of each producer is impossible.

Nowadays Smart Home platforms are used for building intelligent home devices. From the end user point of view the Smart Home platform suggests the wide functional capabilities. From the IT point of view Smart Home platform can be conceded as an effective framework or as a set of COTS solutions. For example, data acquisition can be realized on the base of Internet of Things technologies. Nowadays a generalized solution for defining and improvement results of failure for intelligent home devices is to be founded. This can be done by means of usage of automatically generate programs which can realize smart devices functionality.

The systems under consideration by the most part are distributed multitier client-server systems, the client side of which, especially the lowest layer, includes heterogeneous periphery modules with rather weak build-in processors and limited volume of memory.

In the process of supporting of described above class of systems very often one can observe two typical problem situations, which appear in the process of diagnostic.

1. According to the results of group diagnostic of periphery equipment it is detected that status parameters of one or more periphery device is not correct. Group diagnostic is realized on the server side and assumes gathering of limited volume of diagnostic data. But for complete fixing of the error it is necessary to realize a number of specialized diagnostic procedures.

2. The server receives messages about error situations in one or more peripheral modules. Error messages are created in the process of modules self diagnostics. Total number of error situations is limited. Detail information about the error situation as a rule is absent. Because of it, additional diagnostic procedures, adaptive to the observed situation, are to be realized.

Usage of general algorithms for fixing described above situations is not effective and in many real situations is impossible. For fixing modules errors in such situation it is necessary to generate diagnostic procedures taking into account specific features of the situation.

One can meet a number of similar situations, when it is necessary to generate diagnostic procedure in run time.

The general idea of the proposed solution is given in Fig. 1.

So the problem of diagnostic of such kind of systems can be conceded as a problem of synthesis of diagnostic algorithm "on the fly". This algorithm has to gather all necessary information about module status and realize the procedure of error fixing. This problem is to be solved in the limited interval of time in conditions of limited computing recourses. The diagnostic procedure (script) is generated on the server side and generated script is downloaded to the periphery module for execution.

The set of restrictions for the process of diagnostic is defined by the infrastructure and available hardware. The set of restrictions is formed on the base of external restrictions which are formed outside of the system under consideration, parameters of the network infrastructure and the list of diagnostic features, which can be realized on the client side.

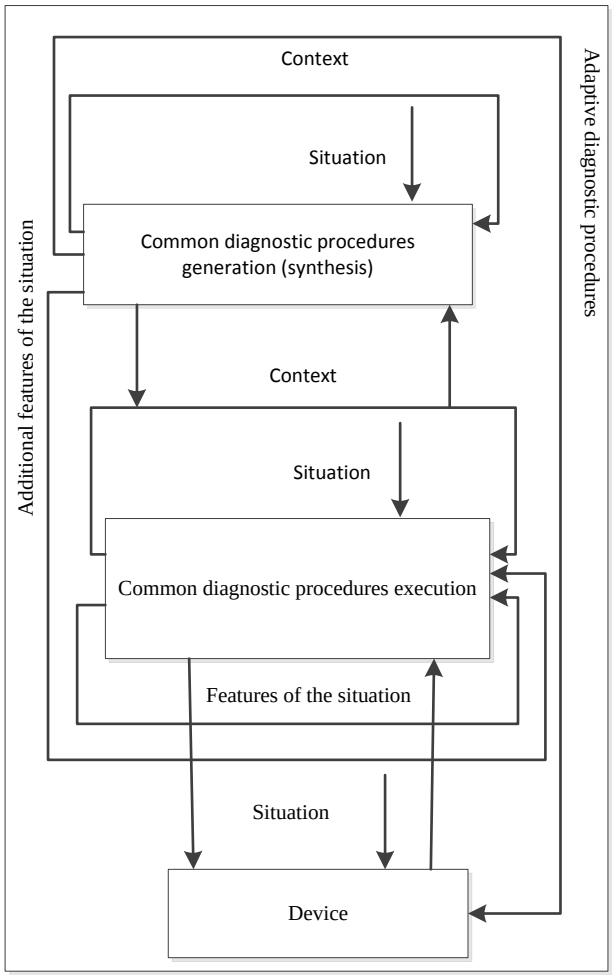

Fig. 1 The general structure of the procedure for devices diagnosis

Diagnostic algorithm must take into account the context. The context is defined by the parameters of the observed status of the periphery modules and user actions which have initiated the error situation.

In order to realize the described above approach the methodology of monitoring algorithms synthesis is to be developed. This methodology must allow generate diagnostic procedures "on the fly" taking into account context and dynamically changing restrictions. The methodology has to include 3 interconnected parts: methodology of identification of presence of a problem, methodology of the problem localization and methodology of fixing of the problem.

It is necessary to mention that nowadays there is no ready to use methodology of such kind.

For solving problems of such kind one can use known methods of program automatic synthesis [1 - 14]. From the point of view of our problem the most perspective are method of resolutions [1 - 2] and reverse inference method [3]. There a number of realizations of these methods [4 - 14]. All realizations assume proving existence of program on the given set of conditions and extraction the program from proving.

Deductive program synthesis [6, 7] assumes that initial data $\{d_s\}$ and target result $\{d_w\}$ are given. Conditions (rules) are defined as

$$F_{zv}(d_{zv_e}; e=\overline{1,E_z}) \to d_{zv_a}; z=\overline{1,Z};\ v=\overline{1,V_z}, \qquad (1)$$

This expression couples initial and final states of the system. Different functions realized by the system may be conceded as $F_{zv}(\cdot)$. In order to solve the problem it is necessary to receive the program which allows migrate from $\{d_s\}$ to $\{d_w\}$. The target program PRG can be defined as:

$$PRG = \begin{Bmatrix} F_{zv}(d_{zv_e}; e=\overline{1,E_z}) \to d_{zv_a}; \\ \{d_s\};\ \{d_w\};\ z=\overline{1,Z};\ v=\overline{1,V_z} \end{Bmatrix} \qquad (2)$$

For successful solving this problem it is necessary to have formal description of the processes in the form of the set of conditions (rules), linking different data. Formal description of simple processes is not very sophisticated problem. But synthesis of reconfigurable programs is much more complex problem, because it is necessary to use special rules describing system structure. Nowadays these rules are not perfect and this problem is to be solved.

It is necessary to mention that known methods of program synthesis are too complex, that limits the scope of their usage. Basically, the known methods are oriented to single-level automatic synthesis of small programs. To reduce the complexity of this synthesis, various methods of parallelizing the processes of proving the existence of programs and extracting them from the output are used. However, these methods only partially smooth out the acuteness of the problem of rapid synthesis of reconfigurable programs for smart devices [6 - 14].

In order to increase the level of intelligence and increase performance of diagnostic and repairing system it necessary to develop modern more effective methods of automatic program synthesis.

When programs are developed manually multilevel approach is used. The idea of step by step program synthesis

was suggested in 1983 [15]. Then, this approach has been developed in other works [16, 17]. But up to present time this idea was not properly formalized and realized. In the present paper an approach to practical implementation of this idea is suggested.

In the section 2 of the paper the formal model of reconfigurable program in terms of relatively finite state operational automata is suggested. In the section 3 a new method of multilevel automatic synthesis of programs is discussed. In the section 4 questions of correctness of results and computational complexity of program synthesis are discussed. Results of multilevel synthesis of diagnostics and repairing programs for digital cable TV (DCTV) and recommendations about suggested approach usage are given in the section 5.

II. Model of Reconfigurable Program

Let us consider formal model of reconfigurable program. It is well known that any program with fixed structure can be described by the final state automate [18]. For formal description of reconfigurable program including self repairing and self reproductive programs it is necessary to take into account a number of additional conditions. These programs can be formalized by means of relative finite state operational automata.

Each automate OKAr in r-th moment of time can be described in terms of 10 parameters,

$$OKA_r = \{\bar{d}_{a_r}, \bar{d}_{b_r}, \bar{d}_{c_r}, F_r^b, F_r^c, DA(\bar{d}_{b_{r-1}}), DB(\bar{d}_{b_{r-1}}), DC(\bar{d}_{b_{r-1}}), FB(\bar{d}_{b_{r-1}}), FC(\bar{d}_{b_{r-1}})\}, \quad (3)$$

where: $\bar{d}_{a_r}$ - input data vector; $\bar{d}_{b_r}$ - vector of internal state parameters; $\bar{d}_{c_r}$ - vector of output state parameters. Functions of transitions F_r^b in (3) define automata transitions from one internal state to another internal state,

$$\bar{d}_{b_{r+1}} = F_r^b(\bar{d}_{a_r}, \bar{d}_{b_r}) \quad (4)$$

Function of states of output F_r^c can be described as

$$\bar{d}_{c_r} = F_r^c(\bar{d}_{a_r}, \bar{d}_{b_r}) \quad (5)$$

States $\bar{d}_{b_r}$, $\bar{d}_{c_r}$, $\bar{d}_{a_r}$, and functions F_r^b, F_r^c, which define automate in r-th moment of time, must satisfy following conditions:

$$\bar{d}_{a_r} \in DA(\bar{d}_{b_{r-1}}) \quad (6)$$

$$\bar{d}_{b_r} \in DB(\bar{d}_{b_{r-1}}) \quad (7)$$

$$\bar{d}_{c_r} \in DC(\bar{d}_{b_{r-1}}) \quad (8)$$

$$F_r^b \in FB(\bar{d}_{b_{r-1}}) \quad (9)$$

$$F_r^c \in FC(\bar{d}_{b_{r-1}}) \quad (10)$$

Condition (6) says, that state of automate (program) in r-th moment of time is limited by the set $DA(\bar{d}_{b_{r-1}})$ allowed states, defined for r-1 moment of time. According to (7) internal state of automate for r-th moment of time must be a member of the set $DB(\bar{d}_{b_{r-1}})$ of allowed internal states. Expression (8) defines limitations for allowed states of automate outputs. These states must be members of the set $DC(\bar{d}_{b_{r-1}})$. According to the condition (9) transition function F_r^b for r-th moment of time must be a member of the set $FB(\bar{d}_{b_{r-1}})$ of allowed functions for r-1 moment of time. The set $FB(\bar{d}_{b_{r-1}})$ of transition functions defines the instruction set of the automate for r-th moment of time. F_r^b is defined by the vector $\bar{d}_{b_r}$, which describes parameters of internal states of automate. According to (10) function of outputs F_r^c of DCTV at r-th moment of time must be a member of the set $FC(\bar{d}_{b_{r-1}})$ of allowed functions, which are active at r-1 moment. Transition from automate OKAr to automate OKAr+1 at r+1 moment of time one can describe as

$F_r^b : OKA_r, \bar{d}_{a_r} \rightarrow OKA_{r+1}$.

This process has following steps: i) definition of the basic sets of allowed parameters of automata, ii) marking these sets by upper index «o», not taking into account their correlations with internal states. Let us define the complex of the basic sets as

$$DOKA^0 = \{DA^0, DB^0, DC^0, FB^0, FC^0\} \quad (11)$$

From elements of these basic sets (11) one can form allowed sets of parameters of higher i-th levels, $DOKA^i = \{DA^i, DB^i, DC^i, FB^i, FC^i\}$. As a result the automate (program) may be characterized by allowed sets of parameters on different hierarchical levels,

$$DOKA^0 \Leftrightarrow DOKA^1 \Leftrightarrow .... \Leftrightarrow DOKA^i \Leftrightarrow .... \Leftrightarrow DOKA^K \quad (12)$$

Taking into account that automate for current moment of time is described by $DOKA^i = \{DA^i, DB^i, DC^i, FB^i, FC^i\}$ this complexes in general case are changing in time. They can be described as a function of its internal state $DOKA^i = DOKA^i(\bar{d}_{b_{r-1}})$.

In a number of cases by means of expending of the set of

automate internal states from (3) - (10) one can exclude functions (5) and correlated with them conditions (8), (10). As a result we receive automate reduced by parameters (4), (6), (7), (9), but with saving ability for reconfiguration.

$$OKA_r^* = \{\bar{d}_{a_r}, \bar{d}_{b_r}, F_r^b, DA(\bar{d}_{b_{r-1}}), DB(\bar{d}_{b_{r-1}}), FB(\bar{d}_{b_{r-1}})\}. \quad (13)$$

While using logical variant of presentation function of automate transition (4) has a view

$$F_r^b(\bar{d}_{a_r}, \bar{d}_{b_r}) \to \bar{d}_{b_{r+1}}. \quad (14)$$

If the set of internal states is expended not only by output states but also with input states then in (14) function $F_r^b(\cdot)$ from $\bar{d}_{a_r}$ maybe not shown.

Distinguishing feature of the described above relatively finite state operational automate is that the set of allowed parameters are true only on one stage (transition) and they are defined relatively to previous state. There is a possibility to change in a full automate not only the set of allowed input, output and internal states, but also the sets of transaction and output functions of automate. In particular cases full automate can be reduced to other automate, with allowed sets of parameters which do not depend upon previous internal states.

Automate (3) – (10) can be considered as a model of a fully reconfigurable program. Also it can be conceded as a model of the system which operates according to this program. Each such automate can be conceded as a complex of coupled automate of lower level. Migration between the levels from the formal point of view can be conceded as a process of tuning of the set of allowed parameters. The length of the record about the same functionality in the form of relatively finite state operational automate depends essentially upon the level of hierarchy used for automate operation description.

III. Program Synthesis Method

Taking into account initial statements (1), (2) and using introduced in Section 2 model (3) – (14), one can formulate the problem of program synthesis in the following form. Initial data $\{d_s\}$ divided into groups $\{d_{s0}\}, \{d_{s1}\}, \ldots, \{d_{si}\}, \ldots, \{d_{sK}\}$ according levels of hierarchy of process description are given. Final result $\{d_w\}$, in the form of groups of data $\{d_{w0}\}, \{d_{w1}\}, \ldots, \{d_{wi}\}, \ldots, \{d_{wK}\}$, correlated with levels of hierarchy are also given. Following conditions are defined

$$F_{zv}^i(d_{zv_e}^i; e = \overline{1, E_z}) \to d_{zv_a}^i; z = \overline{1, Z_i};\ v = \overline{1, V_{iz}};\ i = \overline{1, K}; \quad (15)$$

$$F_{zv}^i(\cdot) \in FB^i; \quad (16)$$

$$d_{si}, d_{zv_e}^i, d_{zv_a}^i, d_{wi} \in D_i; \quad (17)$$

$i = \overline{1, K}$, which couples initial data $\{d_{s0}\}, \{d_{s1}\}, \ldots, \{d_{si}\}, \ldots, \{d_{sK}\}$ with the result $\{d_{w0}\}, \{d_{w1}\}, \ldots, \{d_{wi}\}, \ldots, \{d_{wK}\}$.

Taking into account these conditions it is necessary to generate the program which allows migrate from initial data $\{d_s\}$ to result $\{d_w\}$.

In (15) – (17) following designations are used: $F_{zv}^i(\cdot)$ - functions of z-th kinds и v-th types, which can be realized by a program on i-th level of hierarchy; $d_{zv_e}^i, d_{zv_e}^i$ - data; K - number of levels of hierarchy; FB^i, D_i - the set of allowed functions and the set of allowed data of i-th level.

According to (15) on the i-th level of hierarchy with the help of function $F_{zv}^i(\cdot)$ with known $d_{zv_e}^i$ one can define data $d_{zv_e}^i$. Expressions (16), (17) define restrictions for functions and data on i-th level.

Generalized algorithms of solving the stated problem can be presented as a sequence consisting of 11 steps.

1. Begin $i = K + 1$.
2. $i = i - 1$. Lowering level of hierarchy by 1.
3. If $i < 0$, then ending of problem solving with presenting positive or negative result.
4. Investigating conditions of program solving on i-th level.
5. Proving of existence of the program that allows migrate from $\{d_{si}\}$ to $\{d_{wi}\}$ with productivity not lower than needed for i-th level. In all cases proved and unproved results in all levels of hierarchy are stored. Adding them to initial data and final result on the level -1 relatively to current level.
6. If no proving is found, go to step 2.
7. Extraction of the basic program from results of proving.
8. If there no logical conditions in basic program, then go to step 11.
9. Logical condition processing and receiving subprograms.
10. Linking subprogram and generating a single program of i-th level.
11. Testing of accessibility $\{d_{wi}\}$ using results of program synthesis for levels under investigation. If results are positive, then the programs of different levels are to be linked into result program and finishing of problem solving. In other case go to step 2.

Usage of suggested approach to a program syntheses allows solve $k \le K$ problems enough simply. Synthesis problem is to be solved beginning from the top level.

Low complexity of each problem solving is reasoned by small number of conditions to be analyzed. So, on K-th level rough synthesis of program on big block level is realized. In this case it is not necessary to prove existence of transition from $\{d_{sK}\}$ to $\{d_{wK}\}$.

Mismatch $\{d_{wK}\}$ and inference results on this level can be corrected while working on lower levels of hierarchy. With moving to lower levels, volume of data to be proven is decreasing. It is evident that it is necessary to prove that received results are correct. While realizing multilevel synthesis procedure one can put in line to all data and

conditions predicates which are equal to 1 when condition is true (satisfied) or equal to 0 in opposite case.

For proving existence of productive program on each level one can expand initial data by means of usage condition (15). If multi step extension is used it is necessary to check feasibility of main and solubility of auxiliary (logical) conditions. Main condition is feasible if all its variables are free and arguments are defined. Variables can be conceded as free in 2 cases: i) if they are not coupled by auxiliary conditions, ii) if they are coupled, but are used in common with auxiliary conditions, these conditions are soluble their variables are free. Auxiliary conditions are soluble if their variables are defined.

In the case of initial data extension it is necessary to check them for presence given final results. When all final results are presented in the extended initial data set, then proving of existence of the program is to be finished. Received result includes final program using which one can be migrate from initial data to the final result.

For extraction a program from proving one can use the reverse inference algorithm [3, 6, 7]. Realizing the procedure of reverse inference in mirror mode relatively to direct inference it is possible to receive pure program without extra elements. For processing logical conditions in the basic program its subprograms are generated analogically. If rigorous proof of existence of needed program is absent for the defined conditions, it is necessary to realize the procedures of automatic adding of condition and run problem procedure of program solving again.

IV. Program Productivity and Synthesis Complexity

Presented in section 3 algorithm of the problem solving assumes automatic synthesis of programs on the upper level of hierarchy without rigorous proof of the program existence. The prove is to be taken unto account if program productivity would be not lower then needed. In general case, productivity is defined as ability to receive needed result. Without rigorous proof this result can be received partially or exactly but with assumptions done in the process of inference. Not defined data can be conceded as such assumptions in the process of inference. If in spite of presence of not determined data results of prove are taking into account, then latter can be conceded as unproved final results.

It can be explained with the help of the simple example. Let us assume that on the i-th level of hierarchy initial data $\{d_1^i, d_7^i, d_{12}^i, d_{25}^i\}$ and needed final result $\{d_5^i, d_9^i, d_{18}^i, d_{22}^i, d_{33}^i, d_{51}^i\}$ are given. Functions

$$\left\{\begin{array}{l} F_1^i(d_1^i, d_{12}^i) \to d_5^i \\ F_2^i(d_9^i, d_{12}^i, d_{25}^i) \to d_{33}^i \\ F_3^i(d_5^i, d_7^i, d_{25}^i) \to d_9^i \\ F_4^i(d_1^i, d_5^i, d_{25}^i) \to d_{37}^i \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ F_j^i(d_9^i, d_{25}^i, d_{33}^i) \to d_{18}^i \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ F_m^i(d_{12}^i, d_{33}^i, d_{25}^i) \to d_{22}^i \end{array}\right\},$$

are defined. They couple initial data with final results on the proper level of hierarchy.

It is evident that it is necessary to proof existence of the program, which allows transform initial data to the final results with productivity not lower than needed. If not rigorous proof of existence of program is realized, on these conditions schema of initial data extension by means of usage of functions can be presented in the form

$$\left\{\begin{array}{l} d_1^i \\ d_7^i \\ d_{12}^i \\ d_{25}^i \end{array}\right\} \Rightarrow \left\{\begin{array}{l} F_1^i(d_1^i, d_{12}^i) \to d_5^i \\ F_2^i(d_9^i, d_{12}^i, d_{25}^i) \to d_{33}^i \end{array}\right\} \Rightarrow$$

$$\Rightarrow \left\{\begin{array}{l} d_1^i \\ d_7^i \\ d_{12}^i \\ d_{25}^i \\ d_5^i \\ d_{33}^i \end{array}\right\} \Rightarrow \left\{\begin{array}{l} F_3^i(d_5^i, d_7^i, d_{33}^i) \to d_9^i \\ F_4^i(d_1^i, d_5^i, d_{25}^i) \to d_{37}^i \\ F_j^i(d_9^i, d_{25}^i, d_{33}^i) \to d_{18}^i \\ F_m^i(d_{12}^i, d_{33}^i, d_{25}^i) \to d_{22}^i \end{array}\right\} \Rightarrow \left\{\begin{array}{l} d_1^i \\ d_7^i \\ d_{12}^i \\ d_{25}^i \\ d_5^i \\ d_{33}^i \\ d_{37}^i \\ d_9^i \\ d_{18}^i \\ d_{22}^i \end{array}\right\}$$

According to this scheme of extension of initial data on d_{33}^i are to be accepted, not taking into account the fact that d_9^i в $F_2^i(d_9^i, d_{12}^i, d_{25}^i) \to d_{33}^i$ are not defined. It is necessary to use the rule according to which the ratio of the number of undefined arguments in a function must not exceed the fixed value.

At any extension of initial data while proof of program existence the procedure of checking for presence in them final results is to be realized. If result is received with needed value of productivity, then results of proof are to be taken into account. The checking procedure assumes calculating the ratio of already proofed data to the total number of data and comparison of this ratio with needed value.

One can extract the program from the given scheme with the help of reverse inference using proved data, included in final result. The result is the sequence of functions which can be transformed into the executable code. When such program

extraction mechanism is used function $F_4^i(d_1^i, d_5^i, d_{25}^i) \to d_{37}^i$ can be excluded from consideration as an extra one. If variables would present in such scheme, and they would be coupled with conditions, it would be necessary to check their solvability and generate subprograms.

If it is necessary one can realize logical conditions, branching and cyclic programs with the help of algorithms suggested in [7].

Not proofed during inference data (in our case it is d_9^i and d_{51}^i) are moved lower and are added to final results which are to be received on i-1 level. Even on i-1 level requirements to effectiveness of program to be generated may be strict and proof is rigorous.

Usage of not defined data with not rigorous proof of existence of needed programs is correct when two conditions are satisfied. The first condition is that it must increase certainty of final results. Assumptions in the process of inference, on one hand, can decrease certainty of the final results. On the other hand they can essentially increase certainty of the final results because of increasing of the number of output data.

Usage of only one not defined argument of a function during inference allows receive certain benefits from the point of view of receiving final results. Second condition assumes ability of receiving of rigorous proof of existence of needed program on lower levels of hierarchy.

In comparison with known methods, suggested solution allows essentially decrease time complexity of synthesis and generate more complex programs in automatic mode. The upper boundary of time T_H needed for the program solving for suggested approach can be estimated by the formula

$$T_H \approx c\sum_{i=0}^{K} m_i^2 \le c(\sum_{i=0}^{K} m_i)^2 \qquad (18)$$

where: c – constant coefficient; m_i - number of problem conditions on i-th level. It is necessary to mention, that m_i is essentially less than total number of conditions, used in traditional methods of program synthesis. This assessment is valid, when the number of inference steps for multilevel and single level synthesis are the same.

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 multilevel synthesis of programs,

$$T_L \approx c\sum_{i=0}^{K} \frac{m_i^2}{n_i} \le c\sum_{i=0}^{K} m_i^2 \qquad (19)$$

The average estimate of time T multilevel synthesis with regard to (18), (19) is equal to $T = (T_L + T_H)/2$.

V. Experimental Results

Suggested approach was tested on the problem of diagnostic and equipment repairing DCTV networks. Modern DCTV can include dozens of servers, many thousand or even millions of subscribers with different client side equipment. Usually it is a receiver (set-top box) that contains a TV-tuner input and displays output to a television. In order to receive high quality of service, management centers gather information about client equipment status and realize processing of this information. Management centers execute procedures of video streams and equipment status monitoring in order to estimate correctness of their operation and if needed they realize technical support.

A lot of different problem situations correlated with video streams take place in real DCTV. Analyzing results of monitoring one can detect such problems as image blocking, frozen pictures, fuzziness of image, blinking images, black screen, low level of brightness or contrast, etc. Both hardware and software errors can cause such phenomena. Very often original reasons of such errors are not evident. In this case for adequate situation estimation it is necessary to make additional diagnostics. After it repairing procedures are to be realized. Quality of service and total cost of ownership (TCO) of DCTV depend upon effectiveness of realization described above procedures. Traditional approach for solving such creative problems assumes an active participation of humans. As a rule it takes a lot of time and money.

One of typical error situations is absence of image on TV screen. Instead of an image one can see only "black screen". This error is called "No Video" error. Absence of image can be caused by a lot of reasons, i.e. channel level errors, authorization errors, transport level errors etc. It is impossible to define the reason definitely. For operative image repairing it is necessary to define initial reason of the error appearance. Full information about an error can be received by means of analyzing the status of all components of the software deployed on receivers which are used in the process of image forming. The problem is that is impossible to receive all parameters because there are hundreds of parameters and the bandwidth of LAN as a rule is low.

For detecting the reason of "No Video" error two methods were used: traditional single level method [6, 7] and suggested multilevel method. The priory information about correlation of the error "No Video" with failure in tuning component was used. This information allows limit number of parameters to be analyzed up to parameters of one subsystem. In our example tuning component is described only by 7 parameters.

All these peculiarities, taking into account (15), (17), were formalized by 42 conditions. Among them 12 conditions refer to top (second) level of hierarchy and 30 refer to first level. Programs generated by single level and multi level methods were identical. The result program includes 3 functions (scripts) and supports following features:

1. According to error type it can detect program components and list parameters, which define program component status.
2. It can estimate real and reference values of parameters, which characterize the status of the program components.
3. It can define the parameters which do not satisfy requirements.

While using single step approach on each step 42 conditions

are to be analyzed. While using suggested approach, on the second level on each step only 12 conditions are to be analyzed and up to 30 conditions were analyzed on the first level. So, complexity of calculation was decreased in 1.6 times.

The gain compared with the synthesis conditions of the programs at conditions low level in the example was 2.7 times. In the case synthesis of program on 240 conditions, divided into four level (19) with $n_0 = 1$, $n_1 = 3$, $n_2 = 9$, $n_3 = 27$, the synthesis time was reduced by 11 times.

So one can see from the example given above that multilevel approach allows analyze less quantity of conditions and reduce number of inference steps in comparison with single level one and it has essentially less computational complexity.

The implementation of the proposed approach is based on ontological approach. For describing subject domain ontology supporting SPARQL was used. The ontology was also used for defining repairing instructions, using information about parameters which have values different from reference. For solving this problem logical inference was used. Defined sequence of instruction was used for forming scripts. These scripts were loaded into receiver where they were executed in order to realize diagnostic and repairing.

Architecture of receivers assumes that a special component is integrated in the software stack. The component called virtual machine. It is able to load and to execute generated scripts. For writing scripts domain specific script language is to be used. If it is necessary to control equipment status permanently processes can be run in daemon mode.

VI. Real World Example

The example describes the solution of the "No Video" problem supported in a monitoring of one of the cable TV operators.

Description of problem situation "No Video". Error situations bringing to the "No Video" error one can divide into two groups.

Group 1. Errors of functional components operation.

1.1. Error situations caused by errors of linear channels:

a) channel is unavailable;
b) for SDV (Switched Digital Video) channel the parameter "sdv" is not defined;
c) the "SDV" is defined for the channel which is not a SDV channel.

1.2. Channels description:

a) PAT (Program Association Table) is absent;
b) PAT is defined, but the parameter "mpeg program number" for MPEG channels is not defined;
c) PAT is defined, but PID (PMT) number in the stream is absent

1.3. MPEG4 format is not supported.

2. Error of adding the channel in the list of supported by STB (Set Top Box) channels.

3. Wrong PIN code entered by a user.

Group 2. Errors of receiving transport level data. Error data stream from broadcast channel for demonstrating video.

As an example let us consider the situation 1.2 c), caused by an error in the data stream. In order to fix this error one has to analyze correctness of tuning system operation.

Simulation of an error situation "No Video". Using information about links between the "No Video" error (availability of PAT, absence of PID) and status tuning system components, the scope of parameters to be analyzed can be limited up to parameters presented in the Table 1.

General information about STB status, which is requested for any type of error, is defined by parameters shown in the Table 2. This is minimal set of parameters to be used for solving the problem of monitoring.

The process of the error diagnostic and fixing. The structure of the process is shown in Fig. 2. On the figure operations which are used for solving problems are presented. They are of three different types.

Type 1. Core operations. These operations are required for monitoring system lunching and configuration. The list of core operations includes the following operations:

a) operation " Registration of monitoring system components ",

b) operation " 2. Configuration of resident monitoring processes on STB",

TABLE 1. LIST OF INFORMATION PARAMETERS OF TUNING SYSTEM FOR FIXING "NO VIDEO" ERROR

Parameter	Descripttion
vm_ia_firstQamEqGain	Shows the tuner signal level QAM (Quadrature Amplitude Modulation)
vm_ia_firstQamModulationErrorRatio	Shows error QAM of the tuner
vm_ia_firstQamSignalLevel	Shows the level of the QAM signal
vm_ia_firstQamSignalToNoiseRatio	Shows signal/noise ratio
vm_ia_firstQamCorrected	Shows the number of fixed errors QAM
vm_ia_firstQamUnCorrected	Shows the number of not fixed QAM errors
vm_ia_firstQamFrequencyTuned	Shows the frequency on which the tuner is tuned
vm_ia_firstQamVideoPids	Shows the list of video PIDs in the stream
vm_ia_firstQamAudioPids	Shows the list of audio PIDs in the stream

TABLE 2. PARAMETERS DESCRIBING GENERAL INFORMATION ABOUT STB STATUS

Parameter	Description
Channel number	Chanel number
Source_id	Technical channel identifier, which is used by the tuner for tuning to the selected channel
Lineup_id	Channel net identifier
SDV DCM	Dynamic channel net, which broadcasts using digital video technology
SNR (Signal / Noise Ratio)	Signal/noise ratio, which defines signal quality
PAT/PMT for private	Program Allocation Table / Program Map

source ID	Table for source_id of the channel
SG Number	Service group number for the device

c) operation " 5. Unloading of the monitoring system components".

Type 2. Operations of STB self monitoring. These operations assume regular monitoring of the STB status in order to detect certain types of errors. For this purpose the operation "Realization of STB resident monitoring processes" is used.

Type 3. Operation of realization monitoring programs generated on the server side. These operations are used for realization of the process of gathering information about STB parameters which are to be used for localization and fixing errors. For this purpose the operation " 4. Realization of received from the server monitoring algorithms for fixing error situations" is used.

The example of program code of the process of diagnostic of the "No Video" error which is to be used for solving problems of identification, localization and fixing of problem situations linked with absence of image is presented below.

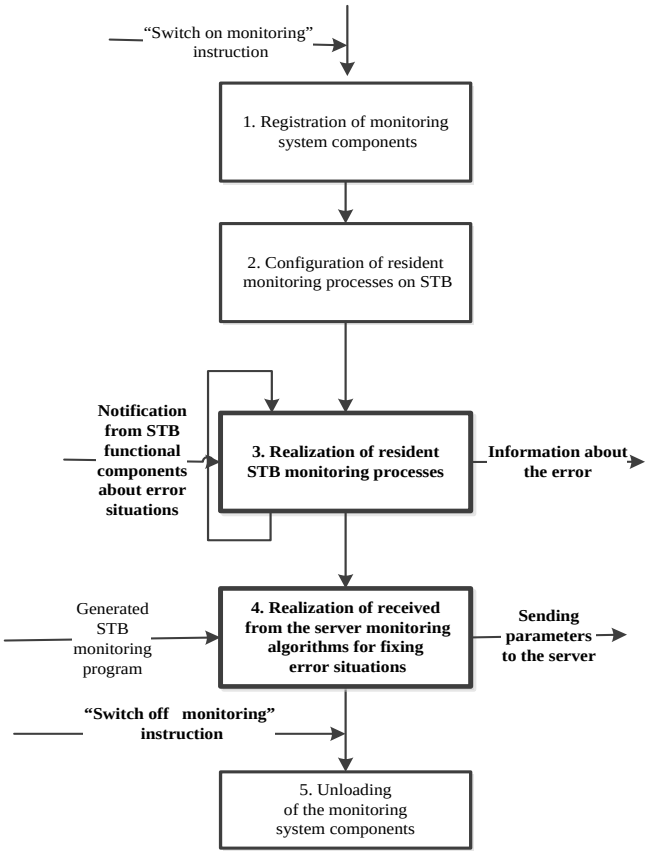

Fig. 2 The process of the error diagnostic and error fixing for "No Video" error

```
/* diagnostic system starting and configuration*/

start()
{
/* diagnostic system components registration */

   register("dc:AppLaunch(int screen, int startTime, int timestampPartMsec, int reason, int channel)", AppLaunch);
   register("dc:AppSessionComplete(int app, int appEndTime, int timestampPartMsec, int sessionId, int reason, int clientSessionId)", AppSessionComplete);
/* configuration of resident diagnostic process*/
   register("dc:NoVideo(int reason)", NoVideo);
}

/* realization of the resident diagnostic process*/

SendAlertCommon(event_id)
{
      reportInt(event_id);
      reportInt(alertTime);
   }
}

/* identification of the error "NoVideo" situation */

SendNoVideo(screen, reason)
{
   no_video_reason =
      screen == -1 ? reason :
      screen == TUNING_FAILED_SCREEN && reason == APP_LAUNCH_REASON ? NO_VIDEO_TUNING_FAILED :
      screen == INSTANT_UPGRADE_SCREEN && reason == APP_LAUNCH_REASON ? NO_VIDEO_NOT_AUTHORIZED :
      screen == PARENTAL_PIN_PROMT_SCREEN && reason == APP_LAUNCH_REASON ? NO_VIDEO_PARENTAL_PIN :
      -1;

   if (no_video_reason == -1)
      return false;
   if (SendAlertCommon(NO_VIDEO_EVENT))
      reportInt(no_video_reason);
   return true;
}

/* Receiving parameters defining STB status for localization of problem situation with the help of algorithm generated on the server side */

SendNoVideoDiag(params)
{
         reportInt(vm_ia_firstQamEqGain);
         reportInt(vm_ia_firstQamModulationErrorRatio);
         reportInt(vm_ia_firstQamSignalLevel);
         reportInt(vm_ia_firstQamSignalToNoiseRatio);
         reportInt(vm_ia_firstQamCorrected);
         reportInt(vm_ia_firstQamUnCorrected);
         reportInt(vm_ia_firstQamFrequencyTuned);
         reportInt(vm_ia_firstQamVideoPids);
         reportInt(vm_ia_firstQamAudioPids);
}
```

VII. Conclusion

As a result of the research, views on multi-level automatic synthesis of reconfigurable programs for smart devices are expanded. A new model of a reconfigurable program is proposed in the form of a relatively finite automation. A new task of multi-level deductive synthesis of programs is mathematically formulated and an algorithm for its solution is developed. Analytic expressions are obtained to evaluate the complexity of multi-level automatic program synthesis.

Suggested model and method form the new approach to formalization of the process of building reconfigurable systems and automatic synthesis of applied programs. Usage of multilevel description of analyzed processes allows divide complex problems into smaller one, for solving which one can find relatively simple solutions.

Realization of descending scheme of synthesis allows find enough quickly needed solutions. Other useful feature of suggested approach is absence of need rigorous proof of existence of solution on the upper levels. Received intermediate solutions can be corrected and on their base one can receive corrected solutions on lower levels. Suggested approach can be effectively used for increasing the level of intelligence of many different information systems that belong to different subject domains.

References

[1] J. A. Robinson. A machine – oriented logic based on resolution principle, Journal of the ACM. 12 (1965) 23 – 41.

[2] C. Chang, R. Lee. Symbolic Logic and Mechanical Theorem Proving, New York: Academic, 1973.

[3] S. Yu. Maslov. Teoria deduktivnykh system i ee primeneniya (Theory of Deductive Systems and Its Applications), Moscow: Radio I Svyaz', 1986.

[4] E. Kh. Tyugu, M. Ya. Kharf. Algorithms for structural synthesis of programs, Programmirovanie. 4 (1980) 3 – 13.

[5] Iskusstvennyi intellect (Artificial Intelligence), vol. 2: Modeli I metody. Spravochnik (Models and Methods. Handbook), Pospelov, D.A. Ed., Moscow: Nauka, 1990.

[6] V. Yu. Osipov. Synthesis of resultative programs to control information and computational resources, Prib. Sist. Upr. 12 (1998) 24 – 27.

[7] V. Yu. Osipov. Automatic Synthesis of Action Programs for Intelligent Robots, Program. Comput. Software. 42 (3) (2016) 155 – 160.

[8] Yu. Korukhova. An approach to automatic deductive synthesis of functional programs, Annals of Mathematics and Artificial Intelligence. 50 (3 – 4) (2007) 255 - 271.

[9] [9] V. B. Novoseltsev. Synthesis of parallel recursive programs in structural functional models, Program. Comput. Software. 33(5) (2007) 293 – 299.

[10] G. Giacomo, F. Patrizi, S. Sardina. Automatic behavior composition synthesis, Artificial Intelligence. 196 (March 2013) 106 -142.

[11] C. Kreitz. Program Synthesis Chapter III.2.5 of Automated Deduction – A Basis for Application. Kluwer Publ (1998) 105 – 134.

[12] A. Avellone, M. Ferrari, P. Miglioli. Synthesis of Programs in Abstract Data Types. LOPSTR 1998: Logic – Based Program Synthesis and Transformation. (1998) 81 – 100.

[13] S. Srivastava, S. Gulwani, J.S. Foster. Template – based program verification and program synthesis/ International Journal of Software Tools for Technology Transfer. 15 (5) 497 – 518.

[14] A. Tahat, A. Ebnenasir. A Hybrid Method for the Verification and Synthesis of Parameterized Self-Stabilizing Protocols. LOPSTR 2014: Logic – Based Program Synthesis and Transformation. (2014) 201 - 218.

[15] E. Kant. On the efficient synthesis of efficient programs, Artificial Intelligence. 20 (3) (May 1983) 253 -305.

[16] W. Bibel, D. Korn, C. Kreitz, F. Kurucz, J. Otten, S. Schmitt, G. Stolpmann. A Multi-level Approach to Program Synthesis, LOPSTR 1998, 1- 27.

[17] P. Fu, E. Komendantskaya. A Type – theoretic Approach to Resolution, LOPSTR 2015, 91 – 106.

[18] F. Wagner, R. Schmuki, T. Wagner, P. Wolstenholme. Modelind Software with Finite State Machines: A Practical Approach, Auerbach Publications, 2006.