Visual Builder of Rules for Spacecraft Onboard Real-Time Knowledge Base

Andrey Tyugashev

Abstract Fault tolerance of spacecraft remains one of the most complex problems in space missions. There are several ways to implement the "onboard intelligence allowing the recovery of a spacecraft in case of abnormal situations caused by hardware or software failures. The most common but inflexible way is "to disperse" the recovery logic in the source code of the flight control software. Our approach implies using onboard real-time knowledge base. The rules of the knowledge base could be added or refined from Earth over the radio channel on a timely basis. Currently, the rules of an onboard knowledge base should be specified in a table form, which entails some misunderstandings in the mission team and consequently leads to errors. The improved approach presented in the paper provides special tools–the visualizer and the visual builder of rules. The approach allows space mission operation engineers without special mathematical or programming background to define, visualize and refine knowledge base rules in a very easy manner. Tools prototypes are currently introduced at JSC Information Satellite Systems, Russia.

Keywords Real-Time onboard knowledge base · Visual builder · Spacecraft control system · Spacecraft's fault tolerance feature · Autonomous control

1 Introduction and Background

1.1 Spacecraft Fault Tolerance

The problem of dependability remains one of the most complex problems in modern space missions. A modern spacecraft has a lot of different onboard systems and equipment (motion control system, power supply system, telemetry system,

A. Tyugashev (✉)
Department of Computer Technologies and Control Systems,
ITMO University, 49 Kronverksky Pr., Saint Petersburg 197101, Russia
e-mail: tau@corp.ifmo.ru

I. Czarnowski et al. (eds.), *Intelligent Decision Technologies 2016*,
Smart Innovation, Systems and Technologies 57,
DOI 10.1007/978-3-319-39627-9_17

thermal control system, etc.). Each system, in turn, consists of dozens of devices, sensors, and aggregates. It is well known that, as a rule, a more complex system is less reliable (except in cases where complexity is specially designed for providing dependability). Thus, it is no wonder that there are many faults and failures during space mission operations. The faults and failures may have a different nature, different levels of damage and methods of recovery. Abnormal situations related to such a complex "system of the systems" can be very tricky and require really smart decision support (in many cases we need reasoning with taking causation relationships into account) to be compensated.

The success of a space mission depends on the possibility to recover the spacecraft in case of hardware or software failures. The recovery of conventional machines exploited on the ground usually involves processes of diagnosis and repair accomplished by a human. If there are cosmonauts onboard, they can use their intellect in an abnormal situation to change or fix the failed equipment. The dramatic history of Apollo 13 is well known as opposed to Salyut 7 rescue expedition [1].

In case of an unmanned mission, the situation is more complicated. To ensure the overall success of space missions, it is necessary to ensure the "fault tolerance" feature of a spacecraft. Fault tolerance is one of the most important components of dependability meaning that even in case of failures a spacecraft remains operational (possibly with some losses in quality). The standard level of the fault tolerance of a spacecraft means parrying of the failure of a single device while fully saving the operational mode [2–5].

Actually, we have a lot of impressing examples of the successful usage of the fault tolerance feature of a spacecraft for the recovery of the operational mode of a spacecraft. How can this feature be implemented?

First, it is reasonable to apply the power of human brain. The computers running intelligent software can assist in this. Probably, we can state that technical diagnostics is one of the conventional domains for the application of knowledge bases and expert systems [6–8]. Spacecraft control processes are implemented by the ground flight control together with the onboard control system [2, 9, 10]. The intelligent decision making support systems are widely used both on Earth and onboard [4]. The only way to utilize the human intelligence to "recover" an unmanned spacecraft is the remote control (there was an exception—the repair of the Hubble telescope [11], but this is just an exception). The remote control mode from the ground control center requires the following. The personnel are engaged in the processes of the analysis of the state of a spacecraft, decision making, transmitting of the required commands onboard. All the operations should be undertaken in real-time mode. It is necessary to ensure the continuous work of the ground complex including telemetry receiving stations, radio command channels, and the infrastructure of a flight control center. Of course, it involves the work of well-prepared personnel who have the excellent knowledge of the operational instructions and are ready to make fast and right decisions. The special methods of learning can be required for training these people [12, 13].

But in many practical cases remote control appears to be fully impossible or too late in time. We mean situations when the radio connection with the ground personnel is impossible due to spacecraft orbits (e.g. low-earth-orbit satellites with a short interval of visibility from each of the ground control points, or probes with a long time of radio signal transmission) [2, 14]. If we face with rapid progress of the abnormal situation onboard, the remote control is useless.

The alternative way for unmanned missions is the use of "onboard intelligence" for autonomous control. The term "onboard intelligence" is enclosed in quotes, and this has a reason behind it. The ways of the implementation of "onboard intelligence" are quite different [15].

Fault tolerance should be envisioned at the initial stage of spacecraft design. De-signers and onboard system engineers make a lot of efforts to support the reliability of the spacecraft equipment. The special measures include the introduction of redundancies. Structural redundancy means duplication (some important units could be doubled or tripled). Functional redundancy means that a system remains operational even if a failure happens (with the downshifted quality, i.e. with less precision, etc.). Spacecraft control logic should be specified in the corresponding obligatory documentation both for normal operations and for abnormal situations. The documents include a list of considered failures with the specification of the level of importance, state diagrams and cyclograms representing the needed reaction [3, 4, 14]. If the failure has been diagnosed, the compensating reaction has to be executed. The reaction implies the reconfiguration of the equipment using "hot" or "cold" reserve or the transition of the spacecraft into one of special modes providing safety (these modes exclude catastrophic consequences).

Then it is necessary to implement the control logic for the considered abnormal situations in spacecraft hardware and software. Nowadays, hardware implementation of onboard control is more a historical issue [9, 16]. The only case when hardware fault tolerance makes sense is the compensation of very fast crashing processes (microseconds or less). In other cases, the common approach is using software [2, 14, 17].

In fact, it is impossible to imagine a space mission without application of computers. Computers are used from the design stage through lifting to Space, operations and support till the end of the lifetime of a spacecraft. Currently, all spacecraft, including micro- and nano-satellites, are equipped with onboard control computer systems which combine several onboard computers integrated into network. The control functions are executed by a special sort of software—onboard software (which can also be called "mission-critical flight control software"). Onboard software consists of hundreds of concurrently running programs [2, 14, 18]. Thus, flight control logic both for normal and abnormal functioning of modern space missions is hidden in the onboard software.

1.2 Ways of the Implementation of the Onboard Intelligence

There are several approaches to introducing control logic into software. The most common but inflexible way is to implement it in the program source code (C, Java, an assembler, etc.). In such a case, any change in control logic should entail a very complex, time-consuming and many-staged process of software re-design, coding and testing (including unit testing, integrity testing, system testing, etc.). When we evaluate labor and time consumption and total costs, the typical proportion between hardware and software of the onboard control system can be characterized as 1:10 [5, 16, 19]. Thus, the total cost of the onboard software lifecycle dramatically grows because of required software maintenance efforts [20]. In aerospace projects (as it was noticed decades ago [10, 16]), the processes of design, development and verification of onboard software became a "critical path" of network scheduling, embracing all works connected with designing and manufacturing of a rocket/space system as a whole [16, 19].

There are a lot of examples of successful implementation of software changes and re-uploading them onboard, even when the distance between the Earth and a deep space probe amounts to millions of kilometers [21–23]. The author knows impressing similar examples in Russian space missions, but they are not published enough in English (for example, the recovery of an Earth observation satellite with the serious change of onboard algorithms and software for operations using another onboard system) [3]. The uploading of onboard software becomes a "routine operation" which has already been performed hundreds of times. Let us consider an example. Jim Erickson, Chief Project Manager of Mars Science Laboratory, states that Curiosity is much more reprogrammable than previous missions. He even called it a "software-defined spacecraft" [21]. The flexibility of the software of Curiosity has sometimes been a problem, because it increases the complexity of the mission. Let us cite Erickson's statement: "The more complicated the software, the more likely you'll not get everything perfect. You'll get surprises. Both in development, test and in operations". But this is a situation where flexibility will help, allowing to redesign the way the rover works in response to a potentially mission-ending hazard that was never anticipated.

A very important issue is that software testing even in theory cannot guarantee a total absence of errors. Moreover, onboard software cannot be fully tested for all possible situations related to the real-time mode of functioning and concurrency [3, 4]. This imperfection reduces the overall effectiveness of space missions.

The dominant trend in modern unmanned space missions is the increase of the planned active lifetime (till 10–15 years) [4, 14]. It is known that onboard electronics faces a growing number of faults caused by the long exposure to cosmic hard radiation. In this case, an abnormal situation emerges, and normal spacecraft operations could be impossible. New kinds of abnormal situations can appear caused by unpredictable flight history and history of failures. They cannot be considered at the stage of designing a spacecraft. The changes in control logic related to these situations should be formulated and implemented at the operational stage.

In summary, there is a need in the tools of prompt correction of spacecraft control logic without the necessity of software re-development and upload. The technologies of such re-engineering of space operations in real time entail issues related to the necessity of a timely reaction to an abnormal situation, providing the safety of a spacecraft, and returning a spacecraft to the operational mode without direct access of human personnel [3, 4].

A much more flexible and promising approach than the implementation of control logic in the source code of a program involves the use of some sort of "intelligent software". It can provide flexibility and reduction of labor and total costs.

Intelligent computer software is a term related to a wide spectrum of applications. The known and well-developed approaches include genetic algorithms, fuzzy logic, neural networks, knowledge bases, and reasoning systems [24–27]. The very impressive results in real-time control, presented in public accessed papers, relate to neural networks [28, 29]. But the use of neural networks requires the process of network training. Training process involves the repeated input of patterns to the network (in some cases hundreds or thousands patterns). Unfortunately, there is a non-zero possibility that the network will never become trained enough.

If we consider the problem domain of space mission control, we should take into account the following factors. This is a typical "mission critical" application, with the commercial and/or military customers. In this case, a principle of "trial-and-error" training is inappropriate. Errors at the stage of operation are not permissible. In practice, only pre-flight training of a neural network can be considered. But in this case we still have the insufficient level of confidence because of the lack of well-defined and documented procedure of decision making. This problem is related to the "dispersion" of decision making rules in the network, which leads to the impersonality in responsibility. However, the issue of personal responsibility is of paramount importance in space missions.

Actually, the principles of neural network training do not fully correspond to the problem domain because of the following circumstance. The number of patterns we can use in the training process is limited. But the typical expected lifetime of a modern spacecraft, as it was stated before, is 10–15 years. Thus, there is a non-zero possibility of the emergence of an abnormal situation which was not used as a pattern. In this case, a neural network can make a serious mistake [14, 30, 31]. Probably, this is a reason why, despite the fact that several companies, including AI Solutions, Interface and Control Systems, and Allied Signal Technical Services, started attempts to introduce intelligence into onboard software in the end of 1980s, the manufacturers of spacecraft are skeptical about this idea.

Meanwhile, during designing and manufacturing of a spacecraft, a team of specialists possesses a sum of certain knowledge about a spacecraft as a controlled object. This theoretically gives an opportunity to formulate unambiguous rules for the compensation of each abnormal situation. Thus, rule-based systems such as knowledge bases and reasoning systems look more suitable than fuzzy logic, neural network and genetic algorithms. A very interesting rule-based approach is

"situational control", which has been developed in the USSR since the end of 1960s by D. Pospelov, Yu. Klykov, L.S. Zagadskaya, and others [32].

We can state that today the main efforts in the problem domain of intelligent autonomous control of spacecraft should be aimed at the creation of tools allowing knowledge acquisition. These tools must utilize the knowledge of spacecraft designers and system engineers and represent it as well-defined rules, provide means for checking a set of rules for completeness and consistency, allow onboard execution of rules in real-time mode. We need a means that would allow the refinement of onboard control rules during the flight. The specified problem is not simple. Today, the information about rules is usually represented in verbal form. These descriptions have a fragmentary nature, lack causation and structural relationships, and conditions for activation. Some fragments of the knowledge remain in a specialist's brain. Formalization, structurization and checking of these rules are very urgent problems.

2 Framework

This work was performed under the contract with JSC Information Satellite Systems, Krasnoyarsk region, Russia. Consequently, one of the required features is that the methods should support "seamless" incorporation into the customer's existing onboard software lifecycle processes. All data formats must be compatible with the customer's existing programming tools and databases.

To date, such advanced and flexible methodology of autonomous intelligent con-trol has been already implemented at customer site. A special onboard real-time interpreter of rules is used for autonomous integrated control of a spacecraft. The interpreter is periodically started by the dispatcher of the onboard operating system at fixed time intervals. The rules are incorporated in the so-called "DKD program" (DKD is the acronym for "Duty Control and Diagnosis" in Russian) [33]. The main functions of DKD autonomous control program are the detection of abnormal situations and the execution of the corresponding set of actions needed to eliminate a failure. Abnormal situations are associated with the patterns of spacecraft state vectors. A state vector consists of elementary conditions reflecting the current onboard situation. We can consider a "general" state vector combining the parameters of all onboard systems (not used in practice), and particular vectors checked at fixed time intervals (for example, a particular state vector can include parameters important in the current spacecraft operation mode).

The presence of particular state vectors allows the implementation of a certain kind of "reasoning". It means that we go from one proposition (predicate, etc.) to another. An additional important feature is that during the processing of one particular state vector we can "activate" checking of another one, if it is necessary.

The DKD program is organized as a set of rules. Each rule combines a state vector and the required actions. Each recognizable abnormal situation is associated

with the pattern of a particular state vector of a satellite. First, we should diagnose the presence of a certain abnormal situation.

Let $A = \{AS_q\}$, $q = 1..S$ is a set of recognized Abnormal Situations.

Let $L = \{\alpha_i\}$, $i = 1..M$ is a set of conditions reflecting current onboard situation.

For example, α_{101} can mean "main gyrodynes are in the operational mode", α_{254}—"the second solar panel is not fully opened", etc.

The full set of conditions forms a "general" satellite state vector. As it was stated above, in practice a more effective way is to use a set SV of particular state vectors (subsets of the general state vector).

Then $SV \rightarrow A$ will be a mapping between patterns of state vectors (in other words, complex conditions like α_{101} ^¬α_{254} ^α_{120}) and Abnormal Situations.

Secondly, a diagnostic program should execute the required set of actions (sup-ported both by onboard equipment and software modules). Suppose $F = K\ U\ P = \{f_j\}$, $j = 1..N$ is a set of actions, which can be executed onboard. F unites set K of commands executed by onboard equipment and the set P of onboard programs.

Similarly, $A \rightarrow 2^F$ will be a mapping between Abnormal Situations and the powerset of Actions (a set of all possible subsets).

But this model is not fully adequate. More precisely, we often need not a single action or just a straight step-by-step consequence of actions, but a "cyclogram" (commonly used term in the aerospace domain), containing pairs (f_j, t_j) where t_j is a time of f_j execution. In other words, a cyclogram represents coordinated synchronized operations.

The specially designed domain-specific language (DSL) is currently used to specify the rules. The language is specially designed to be easily understood by non-programmers and differs significantly from C, Fortran or Java. The rule building is an interactive process supported by a special "REAL" programming system presented in Fig. 1.

Actually, a designer of the control logic fills up the fields of the tables in a special database. The tables are logically connected to represent a structure of the rules. There are a table of onboard parameters, a table of abnormal situations, and a table of available onboard actions. The designer chooses a specific action to fill up the table of "recommendations" associated with the specified abnormal situation.

First, all the parameters checked in the conditions contained in state vectors should be specified, as well as the base of executing actions. Then the user forms a particular state vector and associates a set of actions with it. Then the special programming tool converts the database and rules into compact onboard structures.

The verification of rules is performed using a special testing complex. The main part of this complex is the software for the simulation of functioning of all onboard systems (both for normal modes and for predicted failures) and the physical parameters of the flight of a spacecraft. This testing complex is also used for system testing of all onboard software and for training of spacecraft personnel. The checked rules are saved into the memory of the onboard control system during spacecraft manufacturing. The most important feature is that these data can also be transmitted onboard on a timely basis. Thus, it is possible to change the control logic without the necessity of full re-development of the software. We can see that

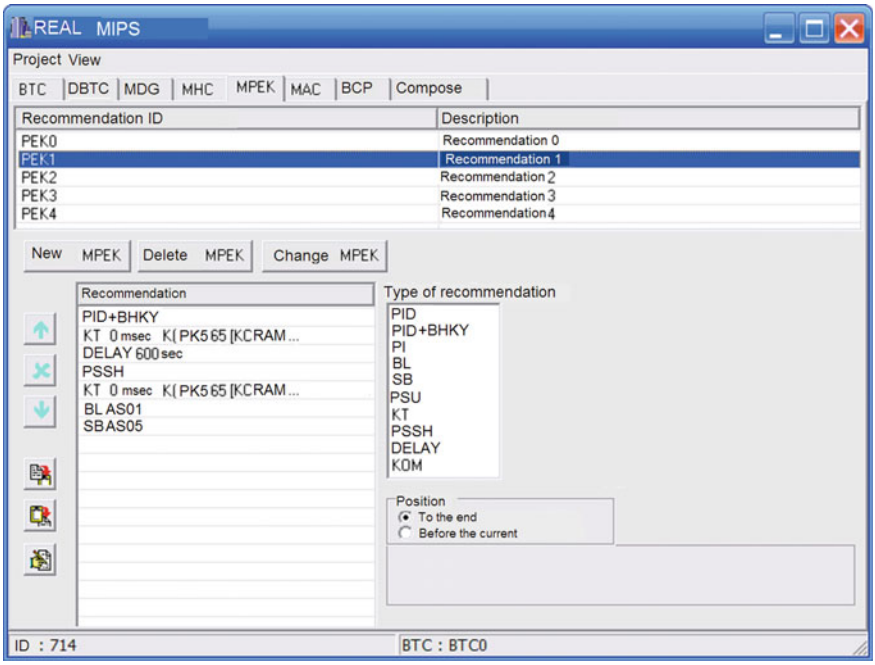

Fig. 1 Screenshot *of "REAL" Programming System*

the onboard rule interpreter is a kind of real-time knowledge base. By the moment of the launch of a spacecraft several hundred rules are usually specified. At the operational stage, this number usually increases by 20–30 % [14]. The opportunity of the specification and updating of rules by non-programmers makes intelligent autonomous control programs the main and most effective tool for spacecraft "remote repair" during its lifetime.

Onboard DKD interpreter supports real-time logical inference (reasoning) about the required actions in forward direction—from the signature of an abnormal situation (the pattern of a particular state vector) to the conclusion about the truthfulness of the hypothesis. Activation of inference is performed in cycle with the individual period for each particular vector, or from one the set of actions activated by one state vector to another. The result of the interactive construction of the autonomous control program is a set of tables printed in a dedicated program document (see Fig. 2).

Despite the fact that the table form of information representation is understood better than a plain text, it is not the best form ensuring clear and fast understanding.

AS #	ABNORMAL SITUATION	PARAMETER	Value	Name of Set of Actions	Actions
0	AS04	СМЕНА	0	РЕК 04	**OFF** AS01, AS03, AS05, AS06 **PIP** (+DBTC1)
1	AS05	СМЕНА	1	РЕК 05	**PIP** (+DBTC1) **BLOCK** AS00, AS01, AS02, AS03 **КТ** 5265 (KLBATC=1), 1,0100 (Start YFCA), 1;5266 (KLBATC=0), 1;5267 (CMEHA=0), **delay** 30 sec. **OFF** AS04, AS06 **РСУ** **BLOCK** AC04 (с №14) **КТ** 5265 (KLBATC =1), 1;0124 (Start YFCB), 1;5266 (KLBATC =0), 1;5267 (CMEHA=0), **delay** 30 sec. **OFF** AS04, AS06 **РСУ** **BLOCK** AS04 (с №14) **КТ** 5265 (KLBATC=1), 1;0100 (Start YFCA), 1;5266 (KLBATC=0), 1;5267 (CMEHA=0), **delay** 30 sec **OFF** AS04, AS06 **РСУ** **PID+ВНКУ** (+DBTC1) **OFF** AS01, AS03, AS04, AS06

Fig. 2 Fragment *of DKD Autonomous Control Program specified by existing DSL*

3 Methods

3.1 Advantages of Visual Form of Information Representation

The main idea of the proposed approach is to combine the flexibility of autonomous satellite control, based on the use of the real-time onboard rule interpreter, and the advantages of the visual form of representation.

There are many reasons for the choice of the visual form of representation of information. Visual form provides simultaneous perception as opposed to textual representation limited by the successive impression. The graphical form of representation is applied to ensure the better reaction time and accuracy of operations of railroad station dispatchers, atomic power station personnel, pilots (the head-up indicator is a good example). The form of information representation is important not only for human-computer interaction, but for human-human interaction, too. In the area of knowledge representation, convenience and clearness for a human is of prime importance. A real-time onboard knowledge base is an example of mission critical systems where the cost of any error or inaccuracy is unacceptably high. The set of rules should be complete, consistent and well structured. The used language makes a “footprint” on the results of thinking. The language should contribute to clear, correct and fast reasoning; the language can be considered as a tool for knowledge [34, 35].

It is a well-known fact that one of the most serious problems of knowledge bases is knowledge acquisition [15, 36–39]. Frequently, a specialist possessing the knowledge is not a mathematician or IT professional. Consequently, he or she faces the problem of the formal representation of the knowledge required by the computer system. A knowledge engineer could help in such a situation, but we cannot fully exclude the "broken phone" effect. There is misunderstanding between the participants of the process. A number of approaches have been proposed for eliminating this problem, for example, the use of a restricted subset of natural language [32, 35]. Another way is an interactive mode of introducing rules. An expert system provides an opportunity to ask clarifying questions. But even in this case we cannot guarantee the absence of inaccuracies and errors. With regard to knowledge, it is reasonable to pay attention to the graphical form of representation.

One can say that the human culture is visually oriented. When we want fast, clear and unambiguous representation, we use graphical form—charts, diagrams, drawing [34]. The best (or even the only) way of representing an enormous amount of knowledge is visual communication. Time tables, bargraphs, maps, even pictorial icons figure prominently in our routine activities [40]. Additionally, the nature of control programs (analysis of logical conditions → detection of the situation → actions) quite corresponds to the graphical form of representation (as opposed to computational programs). Of course, considering the autonomous control domain, we need a means to describe not just descriptive knowledge but also procedural knowledge with "active nature" [15]. The graphics matches the requirements of specification and design stages.

In practice, as a rule, the requirements to the logic of spacecraft functioning are represented in the textual form (or, at best—using tables). Consequently, there are some ambiguities and inconsistencies in the corresponding documents.

Thus, a visual notation and tools for the visual building of onboard real-time knowledge base rules have been proposed.

3.2 Notation

There are a lot of studies devoted to the problem of visual knowledge representation [41–46]. But the performed analysis has shown that any known approach should be very seriously customized if we plan to use it within space missions [35].

The Visual languages for various purposes are being actively developed and used in Russian Aerospace Industry. The examples known to author include Mars Design Bureau, Moscow, Arsenal Design Bureau, Saint Petersburg, Progress Rocket and Space Center, Samara. Unfortunately, the results are practically not published because of many reasons (including security and other issues, see [47]. The very advanced methodology "GRAFIT-FLOKS" with the considering of fundamentals in Human understanding and impression issues was developed and successfully used for years at Academician Pilyugin Center, Moscow [48]. The Visual Notation presented in the paper, substantially based on Parondzhanov ideas.

The notation is not the same, but in some aspects is similar to notation developed at Academician Pilyugin Center. Visual language consists of the following elements:

- Graphical primitives and lines;
- Text labels located inside or outside of graphical primitives.

Actually, the visual notation is based on commonly-used standard flowcharts. The actions are represented by rectangles; the primitives for logical conditions also are intuitively recognizable. But the structure of the flowchart is optimized from the prospective of ergonomics, clear and precise understanding by a human in accordance with the ideas of Parondzhanov [34]. For example, line-crossing is strictly prohibited. The control flow is directed only from top to bottom and from left to right. Straight bottom line from the conditional primitive always corresponds to the "true" branch. These features made the language more concise and thus more intuitive and understandable in comparison with conventional flowcharts.

Some updates have been made in relation to the notation designed by Parondzhanov. First, a flowchart represents one particular state vector (mapping 1:1). A flow-chart consists of several vertical branches which are executed concurrently. The branches correspond to abnormal situations (or it can be said that one branch is one visual rule). Each branch contains exactly one logical condition (complex condition, as a rule), and a set of executed actions. "False" parts are empty. Simple actions are represented by "regular" rectangles. Actions corresponding to satellite control commands with the complex internal structure have the code name displayed in dedicated field, and the comment in other field. Special "KT" block is used to represent a fragment of a "cyclogram" where special fields for specifying of the time of actions are added (the examples can be found in Fig. 3). Delays are

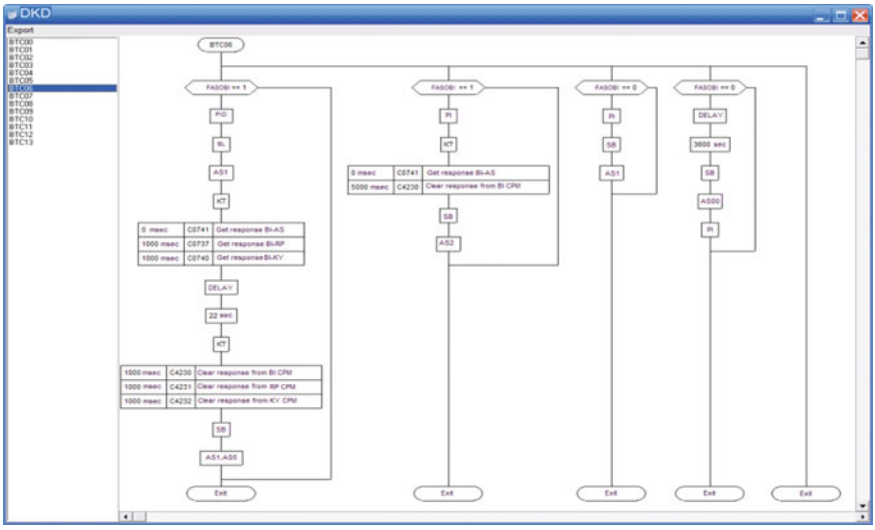

Fig. 3 Screenshot of the Rule's visualization tool

represented by the sequence of two rectangles: first marked as “ПАУЗА” (“Delay” in Russian), and the following rectangle displays the time interval.

4 Practical Results, Discussion and Future Work

One of the causes of errors in mission critical software is the complexity of the development process itself. Misunderstanding between onboard system specialists, de-signers of the satellite control logic, programmers and testers leads to the bugs. In fact, proposed method allows excluding programmers from the development process. This makes it possible to eliminate one type of errors. In practice, we use “programming without programmers” [49]. But other errors caused by the inaccuracies and incompleteness of the rules still preserve and can influence the success of a space mission. Visual verification method [50] is widely used for checking and technical diagnostic of machines and equipment. The structure of the used rules can be visually checked by all the participants of the space mission project. The method of visual rule checking was successfully introduced at the customer site. The developed tools allow:

- Visualization and analysis of previously designed Rules
- Visual building of newly introducing rules.

The screenshot of the Visualization tool is presented in Fig. 3.

As the logical dependencies are allowed between the rules (allowing step-by-step ‘reasoning’), the special feature of the visualizer has been added. We can see and check these dependencies in graphical form as well (see Fig. 4).

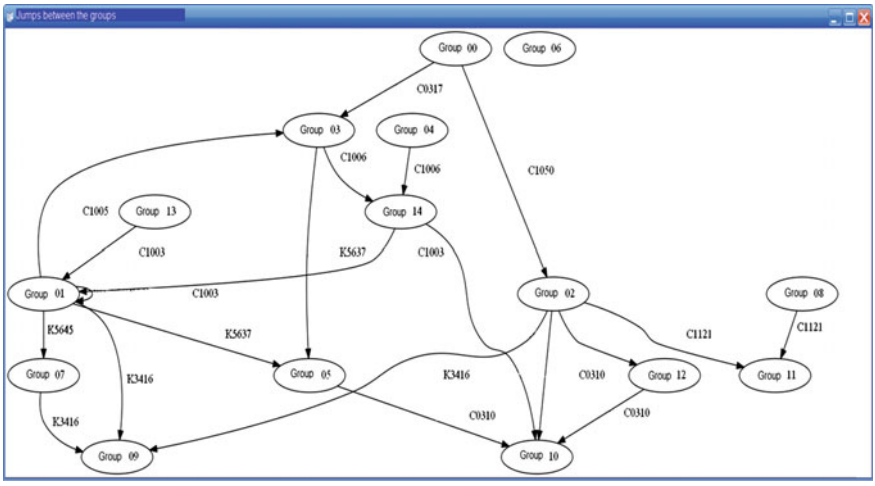

Fig. 4 The screenshot displays the graph of logical dependencies between the rules

The DKD program is represented by the graph; the nodes correspond to rules (state vectors), while the edges display the logical dependencies appearing when during the implementation of one set of actions we find the action that assumes checking of another particular state vector.

Since the designers of the satellite control logic took part in design and discussion of the notation of the visual domain specific language, they enjoy opportunities given by it.

The graphical construction tool supports the creation of an autonomous diagnosis program "from scratch". Initially, the "blank pre-form" of a rule appeared. The user needs to specify the parameters which should be checked in a particular state vector. Then the user can introduce a new abnormal situation and a corresponding set of actions in graphical manner.

As of today, the prototypes of visualization and graphical construction tools have been successfully accepted by the customer. All the tests both at university site and at customer site were executed using real "DKD" programs developed for real satellites which are in use now.

These tools were implemented using C++ programming languages and Graphviz library.

The method and the toolset were documented in accordance with strict industrial level standards applicable to the development of real-time embedded software including DO178B. The presented tools can be classified, by some attributes, as the real-time knowledge base or procedural oriented expert system. Of course, a fully-functional expert system requires additional features to be introduced. It is necessary to support the hierarchy of the rules and introduce more automation into the process of rules verification [14, 51–54].

The presented methods and prototypes of the visualization and graphical construction tools are just first steps in the planned development. The scope of the contract with the JSC Information Satellite Systems covers both the improvement of the developed prototypes and the development of other tools. The planned improvement includes visual language and visual tools for other kinds of autonomous control programs (not diagnostic). Another important aspect is to introduce a feature which will support the detection of incompleteness in the rules of the DKD diagnostic.

The following additional tools are under development now:

- Verification Tool for Satellite Autonomous Integrated Control Programs;
- Documentation Generation Tool providing an automated template-based generation and version control system with a guarantee of the strict correspondence between versions of the program and documentation;
- Networked Integrated Development Environment providing unified access point for designers of Satellite Autonomous Control Programs with saving of user workspaces.

All tools together should form "SIPR MP" (Russian acronym for "System for Intellectual Support for Design and Verification of Integrated Control Programs").

The “SIPR MP” is intended to be used as a complex software engineering toolset. A special stage of the development of “SIPR MP” is dedicated to the collection of notes and comments from users during the introduction of the system at the customer’s site. The found errors should be removed, and the requested improvement of methods and the software engineering toolset should be provided.

5 Conclusions

The paper presents a flexible approach to the fault tolerance control of satellites based on an onboard knowledge base and a real-time interpreter of rules. The architecture of the onboard knowledge base supports the possibility of changing the rules from Earth on a timely basis over a radio channel. The domain specific visual language was introduced for knowledge representation. The visual rule builder provides a clear, user-friendly and unambiguous notation, developed by the designers of the satellite control logic. The process of satellite control is simplified by excluding the necessity of coding the control logic in programming languages and the associated long-term and labor-consuming multi-stage redevelopment cycle of the software. The prototypes of the developed tools were successfully accepted by the customer—JSC Information Satellite Systems, Krasnoyarsk region, Russia (the manufacturer of two-thirds of Russian spacecraft). In future, it is planned to implement additional tools including a verification tool, an automated documentation generation tool, and integrated development environment.

References

1. Ars Technica Information Portal, http://arstechnica.com/science/2014/09/the-little-known-soviet-mission-to-rescue-a-dead-space-station
2. Kozlov, D.I., Anshakov, G.P., Mostovoy, Y.A.: Upravlenie kosmicheskymi apparatami zondirovaniya Zemly: Komputerniye tekhnologii (In Russian). Mashinostroenie, Moscow (1998)
3. Kirilin, A.N., Akhmetov, R.N., Sollogub, A.V., Makarov, V.P.: Metody obespecheniya zhivuchesty nizkoorbitalnykh avtomaticheskykh KA zondirovaniya Zemly (In Russian). Mashinostroenie, Moscow (2010)
4. Akhmetov, R.N., Makarov, V.P., Sollogub, A.V.: Principles of the earth observation satellites control in contingencies. Inf. Control Syst. **1**, 16–22 (2012)
5. Eickhoff, J.: Onboard Computers. Onboard Software and Satellite Operations. An Introduction. Springer, Berlin (2012)
6. Luger, G.F., Stubblefield, W.A.: Artificial Intelligence and the Design of Expert Systems. Benjamin/Cummings Publishing Co, Redwood City, CA (1989)
7. Watanabe, S.: Knowing and Guessing. Wiley, New York (1969)
8. Lambert-Torres, G., Abe, J.M., et al. (eds.): Advances in Technological Applications of Logical and Intelligent Systems: Selected Papers from the Sixth Congress on Logic Applied to Technology. Series Frontiers in Artificial Intelligence and Applications, vol. 186. IOS Press (2008)

9. Tomayko, J.E.: Computers in Space: Journeys with NASA. Alpha Books, Indianapolis, Indiana (1994)
10. Tomayko, J.E.: Computers Take Flight: A History of NASA's Pioneering Digital Fly-By-Wire Project. NASA History Office, Washington, D.C. (2000)
11. Space.com News Portal, http://www.space.com/23640-hubble-space-telescope-repair-anniversary.html
12. Lisitsyna, L., Lyamin, A., Skshidlevsky, A.: Estimation of Student Functional State in Learning Management System by Heart Rate Variability Method. In: Neves-Silva, R., Tsihrintzis, G.A., Uskov, V., Howlett, R.J., Jain, L.C. (eds.) Smart Digital Futures 2014, vol. 262, pp. 726–731. IOS Press (2014)
13. Lisitsyna, L., Lyamin, A.: Approach to Development of Effective E-Learning Courses. In: Neves-Silva, R., Tsihrintzis, G.A., Uskov, V., Howlett, R.J., Jain, L.C. (eds.) Smart Digital Futures 2014, vol. 262, pp. 732–738. IOS Press (2014)
14. Khartov, V.V.: Autonomnoe upravlenie kosmicheskymi apparatami svyazi, retranslyacii i navigacii (In Russian). Aviakosmicheskoe priborostroenie (Aerospace Instrument-Making), 6, 12–23 (2006)
15. Smith, R.K., Muscettola, N.: Knowledge Acquisition for the Onboard Planner of an Autonomous Spacecraft. Technical Report, American Association for Artificial Intelligence WS98-03 (1998)
16. Koczela, L.I., Burnett, G.I.: Advanced Space Missions and Computer Systems. IEEE Trans Aerosp. Electron. Syst. **AES-4**(3), 456–467 (1968)
17. Sghairi, M., de Bonneval, A.: Challenges in building fault-tolerant flight control system for a civil aircraft. IAENG Int. J. Comput. Sci. **35**(4), 120–125 (2008)
18. Koltashev, A.A.: Effectivnaya technologiya upravleniya cyclom zhizni bortovogo programmnogo obespechenia sputnikov svyazi i navigacii (In Russian). Aviakosmicheskoe priborostroenie (Aerospace Instrument-Making), 12, 20–25 (2006)
19. Tyugashev, A.A., Ermakov, I.E., Ilyin, I.I.: Ways to Get More Reliable and Safe Software in Aerospace Industry. In: Program Semantics, Specification and Verification: Theory and Applications (PSSV 2012), pp. 121–129. Nizhni Novgorod, Russia (2012)
20. Kransner, S., Bernard, D.E.: Integrating autonomy technologies into an embedded spacecraft system-flight software system engineering for new millennium. In: IEEE Aerospace Conference, vol. 2, pp. 409–420. IEEE Press, Snowmass (1997)
21. Planetary Society: http://www.planetary.org/blogs/emily-lakdawalla/2014/08190630-curiosity-wheel-damage.html
22. Space.com Information Portal: http://www.space.com/17034-mars-rover-curiosity-software-upgrade.html
23. Planetary Society: http://www.planetary.org/explore/space-topics/space-missions/mer-updates/2004/04-09-mer-update.html
24. Hayes-Roth, B.: An Architecture for Adaptive Intelligent Systems. Artif. Intell. **72**, 329–365 (1995)
25. Nakamatsu, K., Abe, J.M. (eds.): Advances in Logic Based Intelligent Systems: Selected Papers of LAPTEC 2005. IOS Press (2005)
26. Grabot, B., Geneste, L., Dupeux, A.: Experimental design, expert system and neural network approaches: comparison for the choice of parameters. In: International Conference on Systems, Man and Cybernetics 'Systems Engineering in the Service of Humans', vol. 4, pp. 15–20. Le Touquet, France (1993)
27. Nakamatsu, K., Jain, L.C. (eds.): The Handbook on Reasoning-Based Intelligent Systems. World Scientific (2013)
28. Bianchini, M., Maggini, M., Scarselli, F. (eds.): Innovations in Neural Information Paradigms and Applications. Springer-Verlag Berlin Heidelberg (2009)
29. Bianchini, M., Maggini, M., Sarti, L., Scarselli, F.: Recursive neural networks learn to localize faces. Pattern Recognit. Lett. **26–12**, 1885–1895 (2005)
30. Hartmann, G.L.: Fault Tolerant Hardware/Software Architectures for Flight Critical Functions. Introduction/Overview. In: Fault Tolerant Hardware/Software Architectures for Flight Critical

functions. AGARD Lecture Series No.143. NATO Advisory Group for Aerospace Research and Development. Laughton, Essex, UK (1985)
31. Lemos, J.M., Neves-Silva, R., Igreja, J.M.: Adaptive Control of Solar Energy Collector Systems. Springer International Publishing (2014)
32. Pospelov, D.A.: Situational Control: Theory and Practice. Batelle Memorial Institute, Columbus, OH (1986)
33. Kochura, E.V.: Razrabotka macroprogramm integralnogo upravleniya KA (In Russian). Vestnik SibAU **1**, 105–107 (2011)
34. Parondzhanov, V.D.: Druzhelyubnye algoritmy, ponyatnye kazhdomu. Kak uluchshit' rabotu uma bez lishnih hlopot (In Russian). DMK Press, Moscow (2010)
35. Tyugashev, A.A.: Graficheskiye yazyki programmirovania i ih primenenie v sisitemah upravlenia realnogo vremeni (In Russian). Russian Academy of Sciences, Samara, Russia (2009)
36. Shadbolt, N., Schreiber, G. (eds).: Advances in Knowledge Acquisition: 9th European Knowledge Acquisition Workshop, EKAW '96, Springer, New York (1996)
37. Ruiz, P.P., Foguem, B.K., Grabot, B.: Generating knowledge in maintenance from Experience Feedback. Knowl. Based Syst **68**, 4–20 (2014)
38. Osipov, G.S.: Priobretenie znaniy intellectualnymi systemami (In Russian). Nauka, Moscow (1997)
39. Chassiakos, A.P., Vagiotas, P.: A knowledge-based system for maintenance planning of highway concrete bridges. Adv. Eng. Softw. **36**(11–12), 740–749 (2005)
40. Drucker, J.: Graphesis: Visual Forms of Knowledge Production. Harvard University Press, Boston (2014)
41. Chein, M., Mugnier, M.L.: Graph-Based Knowledge Representation: Computational Foundations of Conceptual Graphs. Springer, Berlin (2008)
42. Lengler, R., Eppler, M.: Towards a periodic table of visualization methods for management. In: IASTED Proceedings of the Conference on Graphics and Visualization in Engineering (GVE 2007), pp. 83–88. ACTA Press, USA (2007)
43. Eppler, M.J., Burkhard, R.A.: Visual representations in knowledge management: framework and cases. J. Knowl. Manag. **11**(4), 112–122 (2007)
44. Nobécourt, J., Biébow, B.: Mdws: A modeling language to build a formal ontology in either description logics or conceptual graphs. In: Knowledge Engineering and Knowledge Management Methods, Models, and Tools. LNCS, vol. 1937, pp. 57–64. Springer, Heidelberg (2002)
45. Pfeiffer, H.D., Hartley, H.D.: Visual CP representation of knowledge. In: Bernhard Ganter, Guy W. Mineau (eds). In: 8th International Conference on Conceptual Structures, ICCS 2000. LNCS, vol. 1867, pp.1211–1237. Springer, Heidelberg (2000)
46. Travers, M.: A visual representation for knowledge structures. In: HYPERTEXT'89, pp. 147–158. ACM, NY (1989)
47. Finn, A.: Legal issues for military intelligent decision-making technologies. In: Knowledge-Based Intelligent Information and Engineering Systems. 12th International Conference KES2008. LNCS, vol. 5177, Part I, pp. 14–15. Springer-Verlag Germany (2008)
48. Parondzhanov, V.D., Trunov, Y.V.: Systema upravlenia razgonnogo blocka Fregat (In Russian). Vestnik NPO imeni S.A. Lavochkina (NPO Lavochkina Bulletin), 1 (22), 16–25 (2014)
49. Martin, J.: Application Development without Programmers. Prentice-Hall, PTR Upper Saddle River, NJ, USA (1982)
50. Kalentyev, A.A., Tyugashev, A.A., Bogatov, A.V., Shulyndin, A.V.: Visual toolset for real-time onboard programs verification support. In: Program Semantics, Specification and Verification: Theory and Applications (PSSV 2011), Saint Petersburg, pp. 120–127. Yaroslavl State University, Russia (2011)
51. Sullivan, G.A.: A knowledge-based control architecture with interactive reasoning functions. IEEE Trans. Knowl. Data Eng. **8**(1), 179–183 (1996)

52. Nakamatsu, K., Abe, J.M., Akama, S.: A logical reasoning system of process before-after relation based on a paraconsistent annotated logic program bf-EVALPSN. Int. J. Knowl. Based Intell. Eng. Syst. **15**(3), 145–163 (2011)
53. Giurca, A., Gasevic, D.: Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. Information Science Reference. Hershey, New York (2008)
54. Neves-Silva, R., Rato, L.M., Lemos, J.M.: Time Scaling Internal State Predictive Control of a Solar Plant. IFAC Control Eng. Pract. (Special Issue on IFAC-B'02 Prize Winning Applications), **11**(12), pp. 1459–1467 (2003)