MODELING OF RULES FOR REAL-TIME KNOWLEDGE BASE

A.A. Tiugashev

ITMO University, Saint Petersburg, Russia,

The paper presents approach to modeling of rules for Real-Time Spacecraft's Onboard Knowledge Base with use of specially designed Visual Notation. Visually Checked and Improved Control Rules providing Spacecraft with Fault Tolerance feature can be uploaded onboard in operative manner by radio. As a result, we can reach more efficient and reliable Spacecraft Control. Special software Toolset supporting Visual Notation, including Visualizer of the Rules and Visual Builder, has been developed.

Keywords: Knowledge Modeling, Real-Time Knowledge Base, Visual Notation, Autonomous Control, Fault Tolerance

Introduction

The modern spacecraft/satellite is a technical complex integrating several kinds of systems such as Motion Control System, Power Supply System, Telemetry System, Thermal Control System, etc. Each of system, in turn, consists of devices, sensors, aggregates. Thus, it is not a wonder that many faults, failures and emergencies happens in real Space Missions, caused both by hardware and software errors and bugs.

The cost of such an error is unacceptable high. We can have loss of very expensive satellites, labor results of the thousands of scientists, engineers, technicians (who can represent different countries and participate in a Space Project for years). In a worst case there could be a danger for humans. It is understandable that there are the maximum efforts to avoid these catastrophes.

The possible abnormal situations are a subject of detailed pre-mission engineering analysis, and the Control Logic for the Satellite Recovery in case of analyzed Abnormal Situations should be specified in the corresponding documentation. In manned missions, these documents must be carefully learned by cosmonauts. For an Unmanned Automatic Satellites, the role of Onboard Autonomous Control is extremely important. In case of abnormal situation, Ground Personnel could not have enough time to understand the situation, make right decisions, and transmit correct commands to a Satellite. This is a good reason for advancing in development of autonomous control means.

The Onboard Control System is one of the most complicated Onboard Systems of the Satellite. In case of fault of some onboard device, Control System can analyze the situation and switch to operational backup equipment. Nowadays, even the nanosatellites with the mass less than 10 kilogram and overall dimensions less than one meter, have an Onboard Computer(s) playing the main role in implementation of control processes [1].

Herewith, Satellite Control Logic implemented by Onboard Software. As a result, there is straight dependence between correctness and fault tolerance of the Onboard Software and overall Space Mission's success. Flight (Onboard) Software plays a main role in integration of complex Real-Time Control processes. The modern Onboard Software is a really complicated set of hundreds of concurrently executed software modules, interacting with other programs as

well as with hardware. The links between modules have a different nature, and the very important issue is a Real-Time Mode meaning correct synchronization of the onboard processes.

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 [1]. Thus, the total cost of the onboard software lifecycle dramatically grows because of required software maintenance efforts. In aerospace projects, 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.

There are a lot of examples of successful implementation of software changes and re-uploading onboard, even when the distance between the Earth and a deep space probe amounts to millions of kilometers. 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” [2].

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) [1], [3-5]. 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 [5-8].

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.

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 control 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) [9, 10]. 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 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. Secondly, a diagnostic program should execute the required set of actions (sup-ported both by onboard equipment and software modules). 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 (fj, tj) where tj is a time of action fj 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. 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.

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 [11, 12]. 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 [11].

This is a well-known fact that one of the most serious problems of knowledge bases is knowledge acquisition [13, 14]. 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 [15]. 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. 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 [11].

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” [12]. 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.

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). 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 [16]. 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.

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 [11]. 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 “КТ” block is used to represent a fragment of a “cyclogram” where special fields for specifying of the time of actions are added. Delays are represented by the sequence of two rectangles: first marked as “ПАУЗА” (“Delay” in Russian), and the following rectangle displays the time interval. 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” [17].

Visual verification method 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. 1.

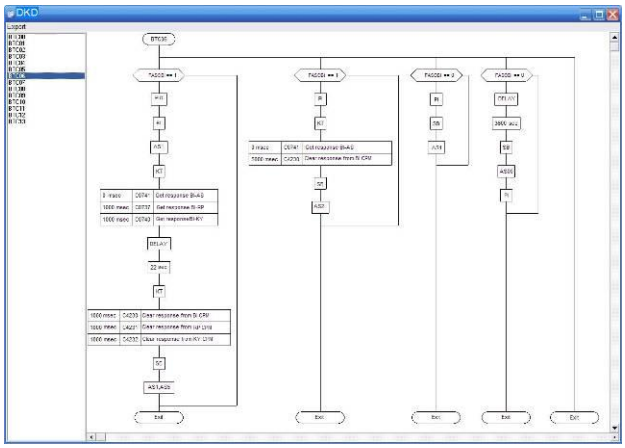

Fig. 1. Screenshot of the Visualization Tool

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.2).

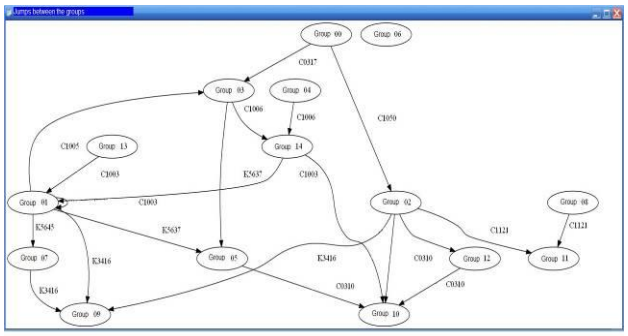

Fig. 2. Links between the Rules

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

Conclusion

The paper presents improvement of flexible approach to fault tolerant control of satellites based on an onboard knowledge base and a real-time interpreter of rules. The domain specific visual language was introduced for modeling of the knowledge reflecting rules of parrying of Abnormal Situations. The visual rule builder provides a clear, user-friendly and unambiguous notation, developed by the designers of the satellite control logic without necessity of programmer’s participation. 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).

References

1. Kransner S, Bernard DE. 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.

2. Planetary Society. Source: <http://www.planetary.org/blogs/emily-lakdawalla/2014/08190630-curiosity-wheel-damage.html>
3. Tomayko, JE. Computers Take Flight: A History of NASA's Pioneering Digital Fly-By-Wire Project. Washington, D.C.: NASA History Office; 2000.
4. Kozlov, DI, Anshakov, GP, Mostovoy, YaA. Upravlenie kosmicheskymi apparatami zondirovaniya Zemly: Komputerniye tekhnologii. [In Russian] Moscow: Mashinostroenie; 1998.
5. Khartov, VV. Autonomnoe upravlenie kosmicheskymi apparatami svyazi, retranslyacii i navigacii [In Russian]. Aviakosmicheskoe priborostroenie (Aerospace Instrument-Making) 2006; 6: 12-23.
6. Kirilin, AN, Akhmetov, RN, Sollogub AV, Makarov, VP. Metody obespecheniya zhivuchesty nizkoorbitalnykh avto-maticheskykh KA zondirovaniya Zemly [In Russian]. Moscow: Mashinostroenie; 2010.
7. Akhmetov, RN, Makarov, VP, Sollogub, AV. Principles of the Earth Observation Satellites Control in Contingencies. Information and Control Systems 2012; 1: 16-22.
8. Tyugashev, AA, Ermakov, IE, Ilyin, II. 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
9. Kochura, EV. Razrabotka macroprogramm integralnogo upravleniya KA [In Russian]. Vestnik SibAU, 2011; 1: 105-107.
10. 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)
11. Parondzhanov, VD. Druzhelyubnye algoritmy, ponyatnye kazhdomu. Kak uluchshit' rabotu uma bez lishnih hlopot [In Russian]. Moscow: DMK Press; 2010.
12. Tyugashev, AA. Graficheskiye yazyki programmirovania i ih primenenie v sisitemah upravlenia realnogo vremeni. [In Russian]. Samara: Samara Scientific Center of Russian Academy of Sciences; 2009.
13. Watanabe, S.: Knowing and Guessing. Wiley, New York (1969)
14. 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)
15. Pospelov, D.A. Situational Control: Theory and Practice. Columbus, OH: Batelle Memorial Institute;1986.
16. Parondzhanov, VD, Trunov, YuV. Systema upravlenia razgonnogo blocka Fregat.[In Russian]. Vestnik NPO imeni S.A. Lavochkina (NPO Lavochkina Bulletin), 2014; 1(22): 16-25.
17. Martin, J. Application Development without Programmers. Upper Saddle River, NJ: Prentice-Hall; 1982.