# Graph-Based and Algebraic Models for Real-Time Flight Control Software

A.A. Tyugashev

*Samara State Transport Iniversity, 443067, 18 1st Bezymyanny Per., Samara, Russia*

**Abstract**

Unfortunately, there are repeating catastrophes in space missions caused by software errors. One of the reasons for it is absence of adequate methods for modeling of real-time control algorithms which are totally distinct from computational (data transformation) algorithms. The purpose of the research is to develop mathematical models suitable for further analysis, design and formal verification of spacecraft's real-time flight control software. The paper presents two models, graph-based and algebraic. These models were successfully used during implementation of CASE toolset for design and verification of real-time spacecraft onboard control algorithms.

*Keywords:* real-time flight control software; graph based model; algebraic model; real-time control algorithm; computer aided software engineering

## 1. Introduction

There are about 500 software modules concurrently running at real-time mode onboard the modern spacecraft, for example, spacecrafts manufactured by Samara Rocket and Space Center 'Progress' [1-4]. These modules have a different nature and objective – system, service, computational, support, etc. The very important part of the onboard flight control software is real-time control algorithms, or so named 'programs for complex functioning' (it means cooperative functioning of various onboard spacecraft's subsystems such as Motion Control System, Telemetry System, Energy Supply System, etc.). The purpose of this part of onboard software is to run needed 'functional' program modules, and execution of the needed commands by particular onboard equipment at 'right' moments of time and proper considering of current onboard situation. It is clear, that the overall success of space missions has a straight dependence on the correct functioning of program for complex functioning. This is an 'evident' example of mission-critical software. So, the cost of errors in such software and algorithms introduced at analysis, design and development phases is too great. The usual way for providing of reliability and quality of flight control software is many-staged testing and debugging with utilization of specially built test beds. This process is very labor and time consuming, but unfortunately cannot guarantee the absence of the errors [3]. The very promising way in this area is application of formal verification methods [3,5]. But unfortunately, the main efforts in area in software formal verification is oriented to computational (data transforming) algorithms and software where adequate mathematical models and methods considering semantics of the algorithms were developed and researched. We can state the inadequacy of these models and methods to nature of real-time control algorithms consisted of not elementary computations but actions related to actuators and other spacecraft's hardware. Accordingly this, the development of the adequate models for this kind of software is very important. The developed models can be utilized in methods of analysis, design and verification which reduce real-time lifecycle labor costs and provide needed level of dependability of real-time control algorithms. In [6] the 'basic' algebraic based mathematical model of real-time control algorithms was presented. This is constructive model, allowing step-by-step building of control algorithms on the basis of 'elementary' actions – so called 'functional tasks', time intervals, and logical conditions. This paper presents some extended models which supplements and clarifies the basic model for further use with various purposes.

## 2. Object of the study

The object of the study is real-time control algorithms coordinating onboard execution of actions needed to achieving of spacecraft's target. It should provide coordinated and well synchronized functioning of onboard spacecraft's systems containing various sensors, actuators, devices, and be a 'conductor of this orchestra'. In case of right synchronization, we will 'hear an adorable harmonic melody', and a sort of 'cacophony' in opposite case.

The very important features distinguishing the control algorithms from the data transformation algorithms are the following. First, we cannot correlate the function (in mathematical understanding), and the control algorithm. And, the correctness of the control algorithm cannot be defined by the contents of the computer memory at the moment of algorithm's end. The correctness of the control algorithm depends on its behavior in all time interval of functioning. Moreover, the values of conditions during execution of data transforming program are totally defined by the input data while the values of the conditions to be considered in control algorithm, are unpredictable because they are formed by the parameters of physical processes in controlled object (for example, velocity of the spacecraft). Actions executed by the control algorithm also can change not only the data in memory of the onboard computer, but influence on the state of the controlled object. When we need, for example, land the spacecraft on the Mars, we need to implement the very complex sequence of the operations with the participation of various onboard devices and mechanisms – but all of them is under control of onboard software.

Unfortunately, the major efforts in the modeling of algorithms were focused on data transformers, since earliest models like Turing and Post Machines, Church's recursive functions and Markov's 'normal algoriphmes'. But we need the adequate models if the control algorithm's semantic, if we want to apply the promising modern methods like formal verification or automated synthesis of the control software with the guarantee of its properness.

As it presented in [6], we can use the following set of tuples ('quads') for representation of semantics of real-time algorithm built from actions executed at particular time if the values of specified logical conditions are equals to 1:

$$UA = \{\langle f_i, t_i, \tau_i, \vec{l_i} \rangle\}, i = 1,..N$$

Each $i$-th quad in the above set describes one action executed by the real-time control algorithm; $N$ is a number of actions executed. Here $f_i$ is an identifier of the action, $t_i$ – starting time of the action, $\tau_i$ - duration. Starting time and duration defined as integers, this is adequate time model in this case because the minimal time difference recognized by the control algorithm is a 'tick' of onboard clock generator. The set of elementary actions $F$ should be previously defined, $f_i \in F$. Logical vector $\vec{l}_i$ specifies the combination of the conditions, allowing action $f_i$ to be executed in time interval $[t_i; t_i + \tau_i]$. For example, logical vector can looks as follows $([\alpha_1=1], [\alpha_2=H],...[\alpha_M=0])$. The 1 and 0 values recognized as true and false, and the third value 'H' means that this condition does not have an impact to execution of the action at specified time. The number of conditions actual for the control algorithm as well as the set $L$ of the condition itself should be settled simultaneously with the set $F$ of actions. We can interpret the logical vector in the model as an analog of the well known 'guard' conditions.

Described model have a very clear and intuitive visual representation looks like Gantt diagram. This was a reason because this model and models inherited from it were intended to use and successfully applied during development of the GRAFKONT/GEOZ integrated development suite for automated design and verification of onboard flight control 'complex functioning' software.

But the 'basic' algebra of the control algorithms had the restricted descriptive power, for instance, there was no possibility to specify arbitrary time intervals between the actions, so it was necessary to introduce 'fictive' actions to taking in account the delays. And the time had the 'relative' nature only; we had no mechanism for binding the actions to the particular moment.

## 3. Methods

### *3.1. Extended algebraic model of the real-time control algorithm*

The proposed model use 'constructive' approach. We can construct new control algorithms from the existing by application of the set of operations. The extended algebraic model contains the following operations:

**Table 1.** Operations of the extended algebraic model of real-time control algorithms

| Name | Mean | Signature |
|---|---|---|
| *CH* | synchronization 'begin-begin' | $(UA_1, UA_2) \to UA$ |
| *CK* | synchronization 'end-end' | $(UA_1, UA_2) \to UA$ |
| $\to$ | direct following | $(UA_1, UA_2) \to UA$ |
| *H* | Overlay | $(UA_1, UA_2, \text{integer}) \to UA$ |
| *3A* | parameterized following | $(UA_1, UA_2, \text{integer}) \to UA$ |
| @ | absolute time binding | $(UA, \text{integer}) \to UA$ |
| $\Rightarrow$ | qualification by the condition | $(\text{condition}, UA) \to UA$ |

'Begin-begin' synchronization applicable to the two control algorithms (denoted in the table above as $UA_1$ and $UA_2$) and forms a control algorithm which includes all quads from both $UA_1$ and $UA_2$, but with the correction of the start time of each action inherited from the $UA_2$. To make the correction, we should calculate overall starting time of the $UA_1$ and $UA_2$. It can be calculated as a minimum of the starting times $t_i$ of the actions included into the UA: $t_{UA} = \min_{i=1..N} t_i$ . After starting times for $UA_1$ and $UA_2$ will be found, we should calculate the difference $\Delta = t_{UA2} - t_{UA1}$. And finally we must add the difference to the all $t_i$ inherited from the $UA_2$. As a result, we will have the control algorithm where all actions from the $UA_2$ will be shifted and the first action from $UA_1$ and $UA_2$ begins at the same time. The *CH* operation is transitive, associative but not communicative.

'End-end' synchronization operation *CK* has the same signature as *CH*, and its result also includes all quads from both arguments. Again, the actions inherited from $UA_2$ should be shifted by $\Delta$. But the rule for calculation of the $\Delta$ is different. The latest action of $UA_2$ in resulting algorithm should ends at the time when ends the latest action of $UA_1$. So, we need calculate overall finish time *et* for $UA_1$ and $UA_2$ as follows: $et_{UA} = \max_{i=1..N}(t_i + \tau_i)$ . And in this case $\Delta = et_{UA1} - et_{UA2}$. The *CK* operation is transitive, associative, but not communicative like *CH*.

Direct following means that in the resulting algorithm the first action inherited from $UA_2$ starts when the latest action inherited from $UA_1$ ends. For this, we need make shift like in the cases above, but the rule is different. We need set the starting time of the earliest action of $UA_2$, as $et_{UA1}$. Difference $\Delta = et_{UA1} - t_{UA2}$ we then need to add to all starting times of actions in resulting algorithm, which are inherited from $UA_2$. In contrast to basic algebra of real-time control algorithms, initially proposed by A.A. Kalentyev, the extended algebraic model includes also the following operations.

Overlay operation *H* is similar to parameterized following *3A* operation. We form the control algorithm including all actions from the arguments $UA_1$ and $UA_2$ like for *CH* and *CK* operations, but applying difference is defined as the additional argument of the operation – integer number. The difference between *H* and *3A* is defined by the following. In the result of H the first action inherited from the $UA_2$ should starts before the end of the latest action inherited from $UA_1$ – so, we deal with 'overlay'. In case of *3A* operation, conversely, the earliest action inherited from the second argument, should starts after the end of the latest action inherited from $UA_1$, and plus one.

Absolute time binding operation allows setting the particular time as the starting time of the control algorithm (starting time of the earliest action). We should find the overall starting time for the existing UA, and then calculate the difference between $t_{UA}$ and the second integer argument of the @ operation. After this, the difference $\Delta$ should be added to all starting times $t_i$ of the all actions to be executed by the algorithm.

Finally, the 'qualification' operation has the UA and the condition as the arguments. We can write $(\alpha_1=0)\Rightarrow UA_1$, and it will mean that in all logical vectors in the UA, we need to update the corresponding component (initially all values for all conditions for all actions settles as '*H*', i.e. action is to be executed imperative).

This model can be successfully utilized for the purposes of specification and verification of the real-time control algorithms. For example, author's applied it at the corresponding stages of flight control software lifecycle for spacecraft [6].

## *3.2. Graph-based model*

The model in previous section specifies the control algorithm at the 'semantic' level. We can see parallel with the denotational semantics of the data transformation algorithms when we nominate the function to be calculated by the program. But the same semantics can be provided by different implementations. Moreover, it is well known fact that in case of software, the quality and characteristics – including efficiency, of the programs with the same semantics, can be quite different.

We need the models for the next degree of detalization. The very popular and effective in practice models in theoretical computer science are graph-based. They are applicable also at the stages of design and analysis of efficiency of implementation of flight control software as well. However, the known models require clarification and extension.

For example, the control flow graph is a very useful and popular model. But initially it describes sequential imperative program executed by the single CPU. Understandably, there are no any essences applicable for describing phenomena of time interval.

But the nature of the complex technical system likes modern spacecraft urgently requires concurrent (multi-tasking) model of computation. It is unsurprising that the onboard software of the modern spacecraft is executed by the control of multi-tasking operating system. Such systems have an application programming interface allowing one process to be started by another (fork). In many cases, onboard real-time operating system allows starting the process with the specified time delay or at absolute time.

This is the reason for defining the extended model – 'timed logical scheme' of the real-time control algorithm. First, we take the well known program control flow graph. In case of control algorithms, nodes with the single outgoing arcs will be associated with any actions executed by the algorithm. Nodes with the two outgoing arcs will be associated with the conditions formed not by the CPU flags only, but related to parameters of spacecraft motion, state of the onboard systems, etc.

We introduce also the special 'weighted' arcs. The weight specified by the integer, will denote the delay before the action associated with the following node be executed. The very important issue is that any 'action' node can have arbitrary number of outgoing 'weighted' arcs additionally to one 'usual' unweighted. The example of the timed logical scheme is presented in Figure 1 in alongside with the visual representation of the corresponding semantic model of control algorithm.

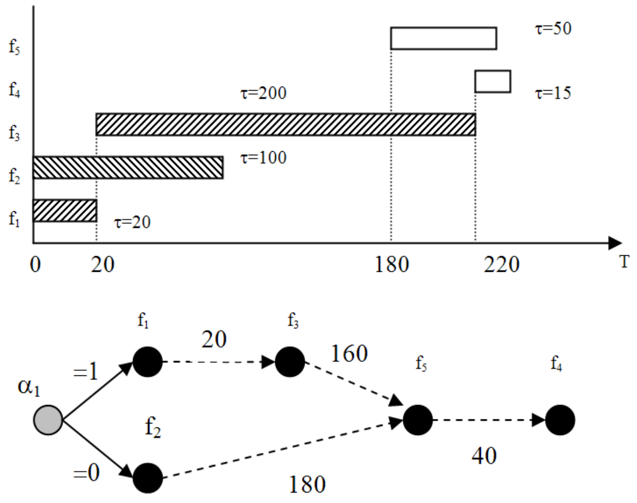

**Fig. 1.** Example of Timed Logical Scheme.

The depicted UA is *{<$f_1$,0,20,($\alpha_1$=1)>, <$f_2$,0,100,($\alpha_1$=0)>, <$f_3$,20,200,($\alpha_1$=1)>, <$f_4$,180,50,($\alpha_1$=H)>, <$f_5$,220,15,($\alpha_1$=H)>}*. Different qualification by the logical vectors can be shown by different texture or color on visual representation of the control algorithm's semantics. Timed logical scheme implements the semantic by the fixation of the 'key' time moments – 0, 20, 180, 200 when the algorithm must perform actions. This 'activations' are divided by relatively time delays: 20, 160, 40, 180 associated with the weighed arcs in the graph-based model. We can see the 'qualified' branches of the timed logical scheme with the corresponding *($\alpha_1$=1)* and *($\alpha_1$=0)* conditions (in this case, there is only the single condition in the logical vector). Then, at timestamp 180, these branches join again.

## 4. Results and Discussion

The presented models looks be much more adequate to the nature of the onboard spacecraft's flight control software than approaches oriented to computational sequential algorithms. Nature of the presented models is quite corresponds to the domain of real-time control of technical complexes consisting of many subsystems, devices, aggregates, etc. The main components of control programs are actions which can be executed both by software modules and equipment. The conditions which influences the process of computation, also does not formed by the input data only, but permanently changing in accordance with the state of controlled object. The semantic model presented in the paper, initially oriented to this particularities.

These features provide possibility to potential application of these models in such area as SCADA systems, power plants, transport, etc. [7].

If we compare presented timed logical scheme of the algorithm with the timed automaton, for example, we will discover that despite there are possibility to describe time related issues, timed automata cannot be used to adequate and unambiguous descriptions of control programs. Timed logical scheme, conversely, initially was developed with this purpose and good corresponds to the factors of problem domain. Moreover, the features of real-time operating systems are taken into account.

## 5. Conclusion

The paper presents two extended models for representation of real-time control algorithms implemented by spacecraft's flight control software. One model is algebraic and another is graph-based. Algebraic model can be applied for 'high level' semantic modeling of the real-time control algorithms. This is important because known models were oriented to computational algorithms and did not take in account the nature of real-time control systems. Semantic models can be used for the accurate and unambiguous specification of flight control software and then applied for formal verification, which is reviewed nowadays as very promising method around the world.

Another presented model is graph-based. It allows analyze the efficiency issues and can be successfully used at the design stage. Constructive nature of these models and their clear visual representations allow developing of GRAFKONT/GEOZ software toolset for specifying and verification of real-time control software. The toolset was introduced at Samara Space Rocket Center.

## Acknowledgements

Author should say many thanks and acknowledgements to President of the Samara University Victor Soifer who granted him a possibility to research and develop the methods and tools for Russian space industry for many years.

## References

[1] Sollogub, A.V. Control of Earth's Remote Sensoring Spacecrafts: Computer Technologies / Kozlov, D.I., Anshakov G.P., Mostovoy Ya.A., Sollogub A.V. – Moscow: Mashinostroenie, 1998.-368 p. (in Russian)

[2] Kirilin, A.N. Spacecrafts Building / Kirilin, A.N., Anshakov G.P., Akhmetov R.N., Storozh D.A. – Samara: Agni Publishing House, 2011. – 280 p. (in Russian)

[3] Tyugashev, A., Ermakov, I., Ilyin I. Ways to get more reliable and safe software in Aerospace Industry // Proc. Program Semantics, Specification and Verification: Theory and Applications (PSSV 2012), July 1-2, 2012 in Nizhni Novgorod, Russia, P. 121-129

[4] Filatov, A.V.,Tkachenko, I.S., Tyugashev A.A., Sopchenko E.V. Structure and algorithms of motion control system's software of the small spacecraft // Proceedings of Information Technology and Nanotechnology (ITNT-2015), CEUR Workshop Proceedings, 2015; 1490: P. 246-251.

[5] Holzmann, G.J. Establishing flight software reliability: testing, model checking, constraint-solving, monitoring and learning / Groce A., Havelund, K., Holzmann, G.J., Joshi, R., Xu, R-G // Annals of Mathematics and Artificial Intelligence, 2014, Vol. 70, No. 4, P. 315-349.

[6] Tyugashev, A.A. Integrated environment for designing real-time control algorithms // Journal of Computer and Systems Sciences International. 2006. Vol. 45. № 2. P. 287-300.

[7] Tyugashev, A. Language and Toolset for Visual Construction of Programs for Intelligent Autonomous Spacecraft Control // IFAC-PapersOnLine Volume 49, Issue 5, (2016) 4th IFAC Conference on Intelligent Control and Automation Sciences ICONS 2016 Reims, France, 1–3 June 2016, P. 120-125