Stochastic Methods for Analysis of Complex Hardware-Software Systems

[1] *A.A. Karnov <karnov@ispras.ru>*
[2] *S.V. Zelenov <zelenov@ispras.ru>*
[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[2] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. In this paper we consider Markov analysis of models of complex software and hardware systems. A Markov analysis tool can be used during verification processes of models of avionics systems. In the introduction we enumerate main advantages and disadvantages of Markov analysis. For example, with Markov analysis, unlike other approaches, such as fault tree analysis and dependency diagram analysis, it is possible to analyze models of systems that are able to recovery. The main drawback of this approach is an exponential growth of models size with number of components in analyzed system. It makes Markov analysis barely used in practice. The other important problem is to develop a new algorithm for translating a model of a system to a model suitable for Markov analysis (Markov chain), since the existing solutions have significant limitations on the architecture of analyzed systems. Next we give a brief description of the context – AADL modeling language with Error Model Annex library, MASIW framework, and also give an explanation of Markov analysis method. In a main section we suggest an algorithm for translating a system model into a Markov chain, partially solving the problem of exponential growth of Markov chain. Then follows a description of further steps, and some heuristics that allow to extremely reduce running time of the algorithm. In this paper we also consider other Markov analysis tools and their features. As a result, we suggest a Markov analysis tool that can be effectively use in practice.

Keywords: Markov analysis; system safety assessment; fault modeling; complex software-hardware system.

DOI: 10.15514/ISPRAS-2016-29(4)-12

For citation: Karnov A.A., Zelenov S.V. Stochastic Methods for Analysis of Complex Hardware-Software Systems. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 4, 2017, pp. 191-202. DOI: 10.15514/ISPRAS-2016-29(4)-12

1. Introduction

In this paper we consider a task related to verification of models of software and hardware systems. Such systems can be, for example, control systems for airplanes, ships, medical equipment, etc. The price of error in these systems is very high, but they are too complicated for ”manually” analysis. Therefore such systems are modeled before implementation. On the stages of design, development, and verification of the models, it is necessary to constantly investigate system safety.

At present, three main methods of system safety assessment [1] are widely used: fault tree analysis, dependency diagram analysis, and Markov analysis. Each method has its own advantages and disadvantages. In this paper, Markov analysis is considered.

Markov analysis works with a Markov chain [2] – a stochastic process, which can be represented as a directed graph with weighted edges. Vertices of Markov chain represent different states, and edges are labeled by probabilities of a transition between states. The main drawback of Markov analysis is a size of Markov chains, which increases exponentially with number of components in the system. In addition, it is necessary to develop an algorithm, that takes system model and translate it to the Markov chain. These problems make Markov analysis not so popular as the other methods, and number of tools that use Markov analysis for complex systems is relatively small. However, such approach has its advantages: Markov analysis allows to look at the entire system, to consider not only causes and probabilities of certain single failure, but also investigate how various failures affect the system in the aggregate. Also Markov analysis, unlike the other approaches, allows to analyze self-recovering systems, since return to operational state is natural for Markov chains.

Thus, the task of development the Markov analysis tool of complex hardware-software systems is quite important and relevant.

2. Context

2.1 AADL and Error Model Annex

Architecture Analysis & Design Language (AADL) [3] is a language, that widely used for describing models of real-time hardware and software systems. Its features include description of both hardware (so-called execution platform) and software components of an analyzed system, and various connections between them. The models, described in AADL, may be used for documentation, for various kinds of analysis and for code generation.

Error Model Annex [4] is an extension of AADL, that allows to simulate appearance and propagation of errors in the system. For each component, a modeller can add a description of component’s behavior states, for example, operational and failed. Transitions between system states are triggered by randomly occured error events and internal errors propagated from other components. An error propagation condition may depend on certain behavior state of the component, some error events, or error propagated from environment. Each propagated error has its own type, that allows to control what is exactly happened in the system. Also transitions between states can be defined implicitly – a state of some component may be a composite state of its subcomponents.

AADL and Error Model Annex together describe not only an architecture, but also error behavior of systems. It becomes possible to evaluate such properties of models as safety, reliability, the availability of its various states and ability to recover from them.

2.2 MASIW

MASIW [5] is an open-source framework for designing and analyzing of integrated modular avionics systems, that use AADL as a modelling language.

The project designed as plugins for Eclipse IDE, includes a variety of tools for working with AADL and Error Model Annex models. There is a big number of different analysis tools, for example, a fault tree analysis tool, but there is no Markov analysis tool.

2.3 Markov analysis

Any model subjected to Markov analysis must be represented as a Markov chain. A Markov chain can be represented in the form of a directed graph with vertices containing system states, and edges labeled with intensities of transitions between corresponding states. A Markov chain has the property of Markov process – a probability of a transition to any state depends only on a current state and a moment in time, and previous transitions are unimportant (can be characterized as memorylessness).

Markov models can be divided into models with discrete and continuous time, as well as time-homogeneous (also called stationary) and time-inhomogeneous. In time-homogeneous Markov chains, the intensities of transitions are constant, while in time-inhomogeneous Markov chains they depend on time. In time-homogeneous Markov chains, transitions occur according to the binomial (or fixed) distribution for discrete-time chains, and according to the Poisson distribution for continuous-time chains.

To determine the behavior of an analyzing system, it is necessary to specify a system of differential equations. The equations follow from the Markov chain. For all Markov processes (and a Markov chain, in particular) we have the Kolmogorov-Chapman equation [6]:

$$P^{(t+dt)}(S_j/S_i) = \sum_{k=1}^{n} P^{(dt)}(S_k/S_i)P^{(t)}(S_j/S_k) \quad (1)$$

This equation means that probability of a transition from state $S_j$to state $S_i$for some time $t + dt$is equal to a sum of probabilities of passes into the target state $S_i$through all of intermediate states S_k.

Consider a time-homogeneous chain with an intensity of the transition between states $S_i$and $S_j$equal to $\lambda(S_i/S_j)$. Then for continuous-time Markov chains, the Kolmogorov-Chapman equation implies a system of differential equations

$$\frac{dP^{(t)}(S_j/S_i)}{dt} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P^{(t)}(S_j/S_i) + \sum_{k=1}^{n} \lambda(S_k/S_i)P^{(t)}(S_j/S_k) \quad (2)$$

And for discrete-time chains, a system of difference equations

$$\frac{P^{(t+\Delta t)}(S_j/S_i) - P^{(t)}(S_j/S_i)}{\Delta t} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P^{(t)}(S_j/S_i) + \sum_{k=1}^{n} \lambda(S_k/S_i)P^{(t)}(S_j/S_k) \quad (3)$$

Denote by S_1a certain initial state of the system, and consider equations (2)-(3) in case when $S_1 = S_j$. Denote by $P_i(t)$the function $P^{(t)}(S_1/S_i)$. Then, the previous equations takes the following form:

$$\frac{dP_i(t)}{dt} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P_i(t) + \sum_{k=1}^{n} \lambda(S_k/S_i)P_k(t) \quad (4)$$

$$\frac{P_i(t+\Delta t) - P_i(t)}{\Delta t} = -\sum_{k=1}^{n} \lambda(S_i/S_k)P_i(t) + \sum_{k=1}^{n} \lambda(S_k/S_i)P_k(t) \quad (5)$$

In addition, initial conditions appear:

$$P_1(0) = 1, P_i(0) = 0, i = -2, n \quad (6)$$

Thus, we obtain the Cauchy problem [7]. The solution of this problem is a set of probabilistic functions of being a system in a definite state. This is the result of Markov analysis.

In this paper, we consider only the analysis of time-homogeneous Markov chains and models, as the most common ones. However, all results can be applied to time-inhomogeneous models, with the only difference being that intensities of Markov chain transitions depend on time, and they need to be stored as formulas, not as numbers.

3. Problem

The goal of this work is a development and implementation of a Markov analysis tool for complex hardware-software systems models within the MASIW framework. The tool takes input of some system model and a set of time points. At the output, the analyzer provides the probabilities of being the system in each of its possible states at moments of time, defined by user.

The main problem is to create a Markov chain on the basis of the original model. First, we need an algorithm that creates a correct Markov chain corresponding to the input data. Secondly, the result chain should be of acceptable size, so that the program can work for acceptable time in limited memory.

After a construction of a Markov chain, further action reduces to solving a Cauchy problem with a system of linear differential equations. An analytical solution of the Cauchy problem is too complicated, resource-intensive, and result is difficult to comprehend, so we use numerical methods.

4. Solution

4.1 Markov chain

The primary task is to translate an AADL model into a Markov chain. In particular, it is necessary to find out what to regard as a chain node and what generates transitions between system states.

Obviously, the node must contain the state of the system, which is a combination of the states of all system components (the states of the components are described in the model as behavior states). However, if we take as a node any of all possible combinations, then the number of nodes will be no less than 2^n, where $n$is the number of system components. Real systems often contain more than 20 components, that, on the one hand, are few, but on the other hand, results in size of such Markov chain outside available memory.

We suggest the following solution of this problem. Let us exclude from the chain all unreachable states of the system, which, as practice shows, are the vast majority. First, some states of the system are unreachable by definition of ananalyzing model. For example, the state of some component may completely depend on the states of its subcomponents. Accordingly, the component can not be in a failed state, while all its subcomponents are in operational states. Second, the failure of some components entails an almost immediate failure of others – for example, a breakdown of a processor entails a failure of all processes running on it. Thus, the state in which the processor is broken, but the processes on it are still working, though reachable in theory, at the very moment of the failure, but instantly replaced by another state.

Thus, we suggest the following approach. We assume that speed of error propagation between components is extremely small in comparison with time of system operation (which, in practice, is the case – for a unit of time measurement usually takes hour and even a day). Let us define a stable state of the sistem as a state, that does not change until new error events occur in the system and its components. We consider as nodes of designed Markov chain only the stable states of the system. The sets of arising events generate transitions between nodes of the chain.

For the sake of saving memory, we insert only reachable states to the Markov chain, and build it dynamically, from the initial state of the system, which is a combination of the initial states of the components. In each new node it is necessary to analyze transitions from the current state of the system. The state can change for some event or combination of events. So, we perform complete search for all possible sets of events – either of them can initiate a new transition. The probability of occurrence of each set of events is easily calculated, since each event contains information about its probability distribution. This is a multiplication product of probabilities of occurrence or negation of occurrence of each of the events, since all events are independent. The total probability of all sets of events, according to the law of total probability, should be equal to 1.

The algorithm is completed when all nodes of Markov chain are analyzed, starting from the node corresponding to the initial state of the system.

```
markovChain.addNode(initialStateNode)
queue.add(initialStateNode)
while not queue.isEmpty() do
    analyzeNode(queue.head())
    queue.add(newNodes)
end while
```

The analysis of each node of Markov chain looks like this: all possible sets of error events are searched, for each of them we calculate a stable state of the system into which the given set leads, and then a new transition (and, if necessary, a new node) is added to the chain.

```
for each errorEventSet in possibleSets do
    state = currentNode.getState()
    repeat
        watchedStates.add(state)
        state = calculateState(state, errorEventSet)
    until watchedStates.contains(state)
    node = markovChain.addNode(state)
    markovChain.addTransition(
        currentNode, node, errorEventSet.getProbability())
    watchedStates.clear()
end for
```

In the above algorithm, the state of the system is considered stable if we have already reached it before. This correctly handles the case when the state of the system has not changed – we have reached the same state as in the previous step. However, in theory, in a self-recovering systems, cycling may occur if an event with a failure and an event with component recovery occur simultaneously. With this condition, the loop stops, but this situation is not handled correctly. One of the main opportunities for further improvement of the algorithm is to improve the condition for achieving a stable state of the system.

4.2 Calculation of new states

In the previous paragraph, a general algorithm for constructing a chain was described, omitting the details of calculating new states of the system. To find out exactly how the system has changed, it is enough to go through all its components, and see what transitions between states are triggered for a given set of events and the current state of the system. The triggered transition is immediately applied to the system, and the algorithm step is completed.

```
for each componentState in systemState do
    for each compositeState in comp.getCompositeStates() do
        if checkStateExpression(compositeState.getExpression())
        then
            systemState.applyTransition(compositeState)
            return
        end if
    end for
    for each transition in comp.getTransitions() do
        if transition.getSource() == compState
        and checkErrorCondition(transition.getCondition())
        then
            systemState.applyTransition(transition)
            return
        end if
    end for
end for
```

The checkStateExpression and checkErrorCondition functions check whether the transition condition is met. Such conditions can be interpreted as a logical formula, where variables corresponding to components behavior states, error events, and propagated errors, have value of true or false, depending on whether the system is in this state, whether an error event has occurred or whether an error of the specified type has propagated.

As soon as some component of the system changes its state, it means that we obtain a new state of the system, and the step of the algorithm is completed. If none of the transitions is triggered, then the system state has not changed, which is noticed by the algorithm described in the previous section.

4.3 Construction and solution of the Cauchy problem

After construction of a Markov chain, the final stage of the Markov analysis of the system is to construct a system of equations and solve the Cauchy problem. As mentioned earlier, each node of the Markov chain generates a differential equation (4) (or similar difference equation (5)). To save memory, it is not necessary to store the system of equations – the equation for any node can be easily constructed dynamically, passing through all transitions entering into this node and outgoing from it.

The resulting Cauchy problem can be solved by a numerical method from the Runge-Kutta [8] family of methods. In the analyzer, two methods are implemented: the Euler method, for discrete-time Markov chains, and the fourth-order Runge-Kutta method, for continuous-time Markov chains. The type of the chain is determined in advance, according to probability distributions of error events. An algorithm for calculating the

variation of the function $P_i(t)$on each time interval delta t, taking into account the dynamic construction of the equation (Euler's method):

```
for each node in markovChain.getNodes() do
    i = indexOf(node)
    res = 0
    for each transition in node.getInTransitions() do
        k = indexOf(transition.getNode())
        res += transition.getProbability() * pPrev[k]
    end for
    for each transition in node.getOurTransitions() do
        res -= transition.getProbability() * pPrev[i]
    end for
    pCur[i] = pPrev[i] + delta t * res
end for
```

Also, the value of the vector of probability functions P (t) is saved at every time point defined by user. As soon as values at each necessary time point are calculated, the algorithm is completed.

4.4 Getting Analysis Results

Since number of system states in Markov chain can be very large, the result of analysis in the form of probabilities of being the system in each of them is practically impossible for reading. Considering that each system has its root component, we group all system states according to the states of the root component.

In this case, all the probability functions within the same group are summed up:

```
for each node in chainNodes do
    i = indexOf(node)
    state = node.getSystemState().get(rootComp)
    analysisResult.get(state) += p[i]
end for
```

After this, for each state of the root component, the probability of being the system in a this state at given time points is obtained. This is the desired result of the Markov analysis of the system.

4.5 Algorithm acceleration

Despite a partial solution of the problem of exponential growth of Markov chain size, the running time of full version of the algorithm still grows exponentially — due to a thorough search of all possible combinations of error events. Thus, we use some heuristics in the final program, accelerating the algorithm.

First, we limit the search of combinations of events. Since the probability of occurrence of one event is usually extremely small, the situation in which several events occur simultaneously is practically impossible. Therefore, a very small numerical parameter, limiting the probability of the combination of events under consideration, was added to the program. If the probability of occurrence of the set of events is less than this parameter, then the effect of the set of events on the system is not considered. This solution significantly reduced the running time of the program, without much loss of accuracy of the result.

The second solution relates to system's ability to self-recovery. In practice, there are few examples of self-recovering systems, and, in most cases, even a short-term failure of the system itself means fatal consequences. Accordingly, if the analyzed system has come to failed state, its further changes are not interesting to us — no matter what else can fail in the already failed system. Therefore, we introduce a set of states of the root component, that are considered as «absolutely» failed. If some node of Markov chain has failed state of the root component, then we do not analyze transitions from it. If analyzed system is not self-recovering, the result of the program remains the same, but is obtained in much shorter time.

Both modifications of the program are optional, as they may change final result in some cases, but their application reduces the operating time by several orders of magnitude. For example, a complete analysis of a system containing 24 components revealed 919 states of the Markov chain and took 1 hour. Limiting the frequency of the events considered by the number 10^{-30}gave a significant gain — the same set of states of the Markov chain and the same result of the analysis were obtained in 7 minutes. Since the system under test was not self-recovering system, the analysis with the stop-on-failed option was correct, and got the same result in 10 seconds. Setting relevant parameters allows to significantly accelerate work of the analyzer. One of the further options for improving the tool can be automatic detection and selection of optimizing parameters.

5. Related works

Markov analysis of AADL and Error Model Annex models is usually applied to systems consisting of only one component. Such algorithms doesnt consider error propagation mechanism and composite states, and limited by root component.

The tool from OSATE [9] framework, created for export AADL model into Markov chain model for PRISM [10] toolset, which provide further steps of Markov analysis, supports only the first nesting level of the component hierarchyand does not support different types of propagated errors. In addition, there were some problems associated with the syntactic correctness of the final PRISM model.

6. Conclusion

In this paper we present a new Markov analysis tool, and in particular, an algorithm for translating AADL and Error Model Annex models into Markov chains. In addition, there were added some improvement for accelerating the algorithm, which make it possible to effectively use the tool in practice.

The presented tool can be further improved in various ways: adding support for time-inhomogeneous Markov chains, accelerating the work of the algorithm, changing some details of algorithm.

References

[1]. "SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Warrendale, USA, Dec. 1996.
[2]. A. N. Shiryaev, Probability (2Nd Ed.). Secaucus, NJ, USA: Springer Verlag New York, Inc., 1995.
[3]. P. H. Feiler and D. P. Gluch, Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, 1st ed. Addison-Wesley Professional, 2012.
[4]. P. Feiler, "SAE AADL error model annex: An overview," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2014. [Online]. Available: https://wiki.sei.cmu.edu/aadl/images/1/13/ErrorModelOverview-Sept222011-phf.pdf
[5]. "MASIW framework," https://forge.ispras.ru/projects/masiw-oss/.
[6]. S. Kuznetsov, "Mathematical models of processes and systems of technical exploitation of avionics as Markov and semi-Markov processes," Civil Aviation High Technologies [Nauchnyi Vestnik MGTU GA], no. 213, pp. 28–33, 2015 (in Russian).
[7]. J. Hadamard, Lectures on Cauchy's Problem in Linear Partial Differential Equations (Dover Phoenix Editions). Dover Publications, 2003.
[8]. U. M. Ascher and L. R. Petzold, Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM: Society for Industrial and Applied Mathematics, 1998.
[9]. J. Delange, P. Feiler, D. Gluch, and J. Hudak, "AADL fault modeling and analysis within an ARP4761 safety assessment," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-020, 2014. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=311884
[10]. M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in Proc. 23rd International Conference on Computer Aided Verification (CAV'11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

Стохастические методы анализа комплексных программно-аппаратных систем

[1] А.А. Карнов <karnov@ispras.ru>
[2] С.В. Зеленов <zelenov@ispras.ru>
[1] Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1.
[2] Институт системного программирования РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В данной работе рассматривается марковский анализ моделей комплексных программно-аппаратных систем. Инструмент марковского анализа может быть использован, в частности, для верификации моделей систем интегральной модульной авионики. Во введении перечисляются основные достоинства и недостатки марковского анализа. К примеру, марковский анализ, в отличие от других подходов — анализа дерева неисправности и анализ алогической схемы, позволяет анализировать модели систем, способных к восстановлению. Основным недостатком данного подхода является экспоненциальный рост размера моделей в зависимости от числа компонентов в анализируемой системе. Это существенно ограничивает возможность применения марковского анализа на практике. Другой важной проблемой является создание нового алгоритма трансляции исходной модели системы в модель, пригодную для марковского анализа (марковскую цепь), так как существующие решения накладывают существенные ограничения на архитектуру анализируемой системы. Далее идет краткое описание контекста, в котором инструмент должен работать — язык моделирования AADL с библиотекой Error Model Annex, набор инструментов MASIW, а также описывается сам метод марковского анализа. В основной части предлагается алгоритм трансляции модели системы в марковскую цепь, частично решающий проблему экспоненциального роста марковской цепи. Затем следует описание дальнейших шагов, а также предлагаются эвристики, позволяющие значительно сократить время работы итоговой программы. В работе также рассматриваются существующие инструменты марковского анализа и их недостатки. В качестве результата данной работы предлагается новый инструмент марковского анализа, который может быть эффективно использован на практике.

Ключевые слова: Марковский анализ; оценка безопасности системы; моделирование неисправностей; комплексные программно-аппаратные системы.

DOI: 10.15514/ISPRAS-2016-29(4)-12

Для цитирования: Карнов А.А., Зеленов С.В. Стохастические методы анализа комплексных программно-аппаратных систем. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 191-202. DOI: 10.15514/ISPRAS-2016-29(4)-12

Список литературы

[1]. “SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,” Warrendale, USA, Dec. 1996.
[2]. Ширяев А.Н. *Вероятность*. Москва: Наука, 1989.
[3]. P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, 1st ed*. Addison-Wesley Professional, 2012.
[4]. P. Feiler, “SAE AADL error model annex: An overview,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2014. [Online]. Available: https://wiki.sei.cmu.edu/aadl/images/1/13/ErrorModelOverview-Sept222011-phf.pdf
[5]. “MASIW framework,” https://forge.ispras.ru/projects/masiw-oss/.
[6]. Кузнецов, С.В. “Математические модели процессов и систем технической эксплуатации авионики как марковские и полумарковские процессы,” Научный Вестник МГТУ ГА, 2015, No 213, стр. 28-33.
[7]. J. Hadamard, *Lectures on Cauchy’s Problem in Linear Partial Differential Equations (Dover Phoenix Editions)*. Dover Publications, 2003.
[8]. U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM: Society for Industrial and Applied Mathematics, 1998.
[9]. J. Delange, P. Feiler, D. Gluch, and J. Hudak, “AADL fault modeling and analysis within an ARP4761 safety assessment,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-020, 2014. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=311884
[10]. M. Kwiatkowska, G. Norman, and D. Parker, “PRISM 4.0: Verification of probabilistic real-time systems,” in Proc. 23rd International Conference on Computer Aided Verification (CAV’11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.