# Automatic Code Generation from Nested Petri nets to Event-based Systems on the Telegram Platform

*D.I. Samokhvalov <disamokhvalov@edu.hse.ru>*
*L.W. Dworzanski <leo@mathtech.ru>*
*National Research University Higher School of Economics,*
*Myasnitskaya st., 20, Moscow, 101000, Russia*

**Abstract.** Nested Petri net formalisms is an extension of coloured Petri net formalism that uses Petri Nets as tokens. The formalism allows creating comprehensive models of multi-agent systems, simulating, verifying and analyzing them in a formal and rigorous way. Multi-agent systems are found in many different fields — from safety critical systems to everyday networks of personal computational devices; and, their presence in the real world in increasing with the increasing number of mobile computational devices. While several methods and tools were developed for modelling and analysis of NP-nets models, the synthesis part of multi-agent systems development via NP-nets is still under active development. The widely used method of automatic generation of target system code from designed and verified formal models ensures obtaining correct systems from correct models. In this paper, we demonstrate how Nested Petri net formalism can be applied to model search-and-rescue coordination systems and automatically generate implementation in the form of the executable code for event-driven systems based on the Telegram platform. We augment the NP-nets models with Action Language annotation, which enables us to link transition firings on the model level to Telegram Bot API calls on the implementation level. The suggested approach is illustrated by the example annotated model of a search and rescue coordination system.

**Keywords:** nested petri nets; telegram bot api; action language; event-based systems; code-generation.

**DOI:** 10.15514/ISPRAS-2016-28(3)-5

**For citation:** Samokhvalov D.I., Dworzanski L.W. Automatic Code Generation from Nested Petri nets to Event-based Systems on the Telegram Platform. *Trudy ISP RAN / Proc. RAS*, vol. 28, issue 3, 2016. pp. 65-84. DOI: 10.15514/ISPRAS-2016-28(3)-5

## *1. Introduction*

Messengers have become the integral part of our life in the recent years; and, almost all the people who have Whatsapp, Viber or Telegram installed on their mobile

devices use them in everyday life. That is all because of hands-on approach in terms of receiving and sending information. Telegram Bot API (TBA)[1] appeared not so long time ago has made a breakthrough in the messengers evolution; and, many IT and business *experts* see the great potential in appliance of the tool for both business and computer science domains.

The variety of TBA usage shows the great diversity of different applied domains starting with service bots, which are designed in order to meet customers requirements, ending with Artificial Intelligence bots (e.g. YandexBot), which can answer different kinds of queries and even strike up and sustain a coherent conversation. The one sphere where TBA could be applied in — people coordinating in different types of special operations. These operations turn out to be extremely difficult to plan and support when it comes to coordination of big squads; especially, in the state of emergency cases. A thorough planning of search and rescue or military operations is rather struggling to deal with, because of the lack of time to create a detailed schedule of part-taking for each agent and deprecated methods for sending and receiving notifications from the agents who are involved in such operations. TBA provides a great opportunity for that purpose because it is extremely easy to use when the bot logic is designed according to a consecutive and well-structured scheme. However, it is not easy to create a coherent TBA logic, because it requires programming skills and is time-consuming. As the time factor plays a crucial role, this makes such system much less attractive and unsuitable in the fast changing context of emergency and rescue operations. Nested Petri Nets (NP-nets) are a well-known formalism which provides an approach for modelling multi-agent systems [2], [3], [4], [5]. NP-nets are generally used to describe the complex processes with dynamic hierarchical structure. NP-nets are convenient for specification of that kind of processes because of the visible and coherent structure [6]. A number of methods for the analysis and verification of NP-nets were developed [7], [8], [9]. However the practical application is impeded by the necessity of manual implementation of the constructed model. Even if the model correctness is verified, code defects can be introduced on the error-prone implementation phase of software construction process. The reasons for such defects: different understanding of the model by a software architect and software developers; the complex behaviour of multi-agent systems with dynamic structure; the distributed systems testing and debugging problems. The alternative to manual coding is automatic code generation from a model to the executable implementation of the modelled system. Automatic generation provide considerable saving of the project resources, reproducible quality of the generated code, better support for round-trip development by regenerating code after model changes. The approach does not guarantee zero-defect implementation, but, after long term usage, a code generation system becomes reliable and allows to obtain code with reproducible quality.

The goal of the project is to develop a code generation system which allows to automatically construct multi-agent systems on the Telegram platform from NP-nets

models. The generated software is designed according to the event-base paradigm and consists of a complex Telegram Bot and mobile Telegram applications. The main purpose of the Telegram bot is to coordinate and communicate with the agents according to the original NP-net model.

The section 2 contains basic notation and definitions. In the section 3, a motivating example of Search and Rescue coordination system modelled with the NP-nets formalism is given. In the section 4, we provide the architecture and technical details on the implementation of the automatic code generation. The section 5 contains the suggested Action Language description. In the section 6, we discuss the application of the suggested technology to the motivating example. The section 7 concerns the related work, the previous studies on NP-nets translations, and further directions.

## *2. Preliminaries*

At first, we provide the classical definition of a Petri Net. A *Petri net (P/T-net)* is a 4-tuple $(P, T, F, W)$ where

- $P$ and $T$ are disjoint finite sets of *places* and *transitions*, respectively;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*;
- $W: F \to N \backslash \{0\}$ – an *arc multiplicity function*, that is, a function which assigns every arc a positive integer called an *arc multiplicity*.

A *marking* of a Petri net $(P, T, F, W)$ is a multiset over $P$, i.e. a mapping $M: P \to N$. By $M(N)$ we denote the set of all markings of the P/T-net $N$.

We say that a transition $t$ in the P/T-net $N = (P, T, F, W)$ is *active* in a marking $M$ if for every $p \in \{p | (p,t) \in F\}: M(p) \geq W(p,t)$. An active transition may *fire*, resulting in a marking $M'$; such that, for all $p \in P: M'(p) = M(p) - W(p,t)$ if $p \in \{p | (p,t) \in F\}$, and $M'(p) = M(p)$ otherwise.

For simplicity, we consider here only two-level NP-nets, where net tokens are classical Petri nets.

A *nested Petri net* is a tuple $NPN = (Atom, Expr, Lab, SN, (EN_1, \dots, EN_k))$, where

- $Atom = Var \cup Con$ – a set of atoms;
- $Lab$ is a set of transition labels;
- $(EN_1, \dots, EN_k)$, where $k \geq 1$ – a finite collection of P/T-nets, called element nets;
- $SN = (P_{SN}, T_{SN}, F_{SN}, \nu, W, \Lambda)$ is a high-level Petri net where
  - $P_{SN}$ and $T_{SN}$ are disjoint finite sets of *system places* and *system transitions* respectively;
  - $F_{SN} \subseteq (P_{SN} \times T_{SN}) \cup (T_{SN} \times P_{SN})$ is the set of *system arcs*;
  - $\nu: P_{SN} \to \{EN_1, \dots, EN_k\}$ is a *place typing* function;
  - $W: F_{SN} \to Expr$ is an *arc labelling* function, where $Expr$ is the

*arc expression language*;

- $\Lambda: T_{SN} \to Lab \cup \{\tau\}$ is a *transition labelling* function, $\tau$ is the special "silent" label.

Let $Con$ be a set of *constants* interpreted over $A = A_{net} \cup \{°\}$; and, $A_{net} = \{(EN, m) | \exists i = 1, \dots, k: EN = EN_i, m \in (EN_i)\}$ is a set of marked element nets. Let $Var$ be a set of *variables*. Then the expressions of $Expr$ are multisets over $Var \cup Con$. The arc labelling function $W$ is restricted such that: constants or multiple instances of the same variable are not allowed in input arc expressions of transitions; constants and variables in the output arc expressions correspond to the types of output places; and, each variable in an output arc expression of a transition occurs in one of the input arc expressions of the transition.

A marking $M$ of an NP-net $NPN = (Atom, Expr, Lab, SN, (EN_1, \dots, EN_k))$ is a function mapping each $p \in P_{SN}$ to a multiset $M(p)$ over $A$. The set of all markings of an NP-net $NPN$ is denoted by $M(NPN)$ Let $Vars(e)$ denote a set of variables in an expression $e \in Expr$ For each $t \in T_{SN}$ we define $W(t) = \{W(x, y) | (x, y) \in F_{SN} \wedge (x = t \vee y = t)\}$ – all expressions labelling arcs incident to t. A *binding* b of a transition $t$ is a function $b: Vars(W(t)) \to A$, mapping every variable in the $t$-incident arcs expressions to a token. We say that a transition $t$ is *active* in a binding $b$ iff: $\forall p \in \{p | (p, t) \in F_{SN}\}: b(W(p, t)) \subseteq M(p)$. An active transition $t$ may *fire* yielding a new marking $M'(p) = M(p) - b\big(W(t, p)\big) + b(W(t, p))$ for each $p \in P_{SN}$ (denoted as $M \xrightarrow{t[b]} M'$).

A behaviour of an NP-net consists of three kinds of steps. A *system-autonomous step* is the firing of a transition, labelled with $\tau$, in the system net without changing the internal markings of the involved tokens. An element-autonomous step is a transition firing in one of the element nets according to the standard firing rules for P/T-nets. An autonomous step in a net token changes only this token inner marking. An autonomous step in a system net can move, copy, generate, or remove tokens involved in the step, but doesn't change their inner markings.

A *(vertical) synchronization step* is a simultaneous firing of a transition labelled with some $\lambda \in Lab$ in a system net with firings of transitions labelled with the same $\lambda$ in all consumed net tokens involved in the system net transition firing. For further details see [5]. Note, however, that here we consider a typed variant of NP-nets, when a type of an element net is instantiated to each place.

## *3. Motivating example*

Search and rescue operations is what happens all over the world; they require the well-trained and skilled employees, well-structured planning, and knowledgeable human management. There were 2447 emergency callouts registered in Russia throughout 2005–2014 [10], and about 100 times more in USA [11]. Earthquakes,

water floods, and hurricanes hit the earth rarely than ordinary emergency cases like fires or gas leaks, but they leave whole regions and even countries devastated, thousands of people killed or lost without a trace. Therefore, the crucial goal of rescuers is to treat such cases quickly and cohesively.

In this example, we will explain how a particular search and rescue operation in an earthquake could be handled with a multi-agent model based on the nested Petri net formalism. First, we need to introduce the purposes of the basic components which we will use further to design our search and rescue coordination plan. Our model consists of the following basic components:

- **System net** – the main component of an NP-net which is a high level Petri net. It will be used to define the activity coordination of the agents involved in the operation. The system net will be implemented as a bot on the Telegram platform to receive the notifications from agents and to process them with the Action Language (AL) event handlers assigned to the transitions of the system net;
- **Element net** – represents the activity of a particular agent type that is supposed to be performed by an agent while taking part in the operation. There are two element nets in our example. The first one corresponds to the acting plan for medical workers involved in the operation; the second one will provide the plan for the rescuers participating in the operation.

The system net in Fig.1 represents the main model of our operation. Basically, it reflects the dependence of the agent actions on server responses. In other words, it describes how an operation coordinator interacts with the rescuers and medics and reacts on their signals to the server. The model deals only with those agent requests where coordinators answer is essential for the further operation progress. An actions happen when the particular agent reaches the state and the coordinator response expected is defined with AL code assigned to the system net transition. To understand how the model works, we need to understand how the agents intercommunicate with the operation coordinating server.

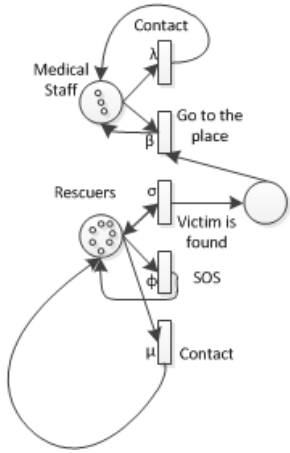


*Fig. 1: The search and rescue system net example.*

In the initial marking of the system net places “Medical Staff” and “Rescuers”, there are all the agents – rescuers and medical staff respectively. The transitions have the following functions:

- Transitions T0 and T4 **”Contact”** — handle communication between the rescuers and the coordinator;
- Transition T1 **”Go to the place”** — represents the event when a rescuer has found a victim, and a medical agent is supposed to go to the place where the victim is found;
- Transition T2 **”Victim is found”** – represents the event when a rescuer agent has found a victim. It precedes the T1 event, as the medical agent needs to start acting only when the victim is found by the rescuer;
- Transition T3 **”SOS”** – a rescuer has stuck in emergency;

The agents behaviour is determined by two element nets. The medical staff element net is depicted in Fig. 3; and, the rescuer element net — in Fig. 5.1. In the real Search and Rescue operations there are usually more element nets and they are more detailed.

*Medical staff element net* represents what kind of actions should a medical agent performs while taking part in the operation. At first, the medical agent needs to get the medicine and learn about the operation. He will not be allowed to the next stage of the operation before he performs both of these actions. After doing that, he is supposed to wait until he receives the notification on the accidence. Then he has to send his arrival time, and start making his way to the place where the accidence had happened. The next two steps are to report the victim condition and to transport the victim to the infirmary. The medical agent also may contact coordinator at any time.

*Rescuers element net* is the model of part-taking for rescuers. Before entering the operation, each agent is required to do the following: get the equipment; obtain the

information about other agents; and, get briefing about the operation. The equipment consists of three parts; and, the agent must equip them all. After entering the operation, the agent has to go to the exploration area. If a victim is found, the agent is supposed to send a photo, a description, and the accurate coordinates of the victim location. If something goes wrong, the agent can just send the location and the coordinator will handle it. Once the exploration is completed, he can receive the coordinates of the new area to explore.

## *4. Architecture*

The way this system is designed relies on three basic components:

- NPNtool (Eclipse plugin) [7] for creating Nested Petri Nets models and linking AL code to the transitions. The main purpose of this tool is to model a system net and element nets which will represent the model of the bot. The AL code will be linked to the transitions and then compiled to the executable file according to the model;
- AL Java-library consists of the AL-compiler and the AL-linker. AL-linker traces the system and element nets, collects all the code from the transitions, and eventually converts in to text files that will be compiled by the AL-compiler. AL-compiler is created with the ANTLR[12] tool. AL-compiler gets an input text file and translates it to the executable artifact that actually represents the Telegram bot;
- Telegram Bot API library that consists of the code for requesting data via HTTP-requests from the Telegram Bot API server.

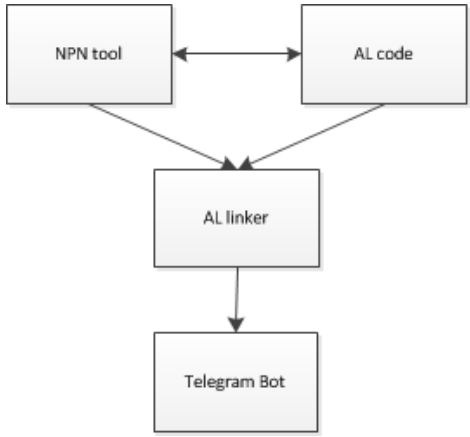


*Fig. 2: The code generation scheme*

The overall technology chain is as follows. At first, a developer creates and verifies the NP-net model of a system via NPNtool. When the model is constructed, the developer inscribes AL code to the transitions according to the expected logic of the Bot. Then, the developer launches AL-linker which traverses the constructed NP-net collecting the textual representation of the transitions AL code into a text file. After that, AL-compiler reads the artifacts generated by AL-linker and generates a Telegram Bot code. The codegeneration of distributed systems from NP-nets models has been studied in [13]. Once a JAR file is compiled from that code, it could be executed. All the actions of the agents are displayed on the Bot host and could be processed at real-time (saved or directly answered) by the coordinator who ran the Bot.

Telegram bot consists of TBA library and several Java classes. Each Java class corresponds to an element net or a system net and stores a number of methods corresponding to the transitions with AL-code inscribed to them. These methods will use TBA to interchange the information. There is also a class that links all the element and system nets libraries together and proceeds the logic using event-based paradigm and asynchronous requests.

It shall be noticed that the compiled Telegram Bot is a server that communicates with the software clients — the rescue and medical staff software mobile clients. The bot is connected to the Telegram server via the webhook technology; namely, all the requests that agents send to the Telegram server via Telegram mobile applications are redirected to and served on the deployed bot server.

The fragment of code in Table 1 represents the method which corresponds to one of the Medical Staff element net transition:

Table 1. Medical staff victim condition report action.

```
public void taskReportVictimsCondition(String
    mes, String chatId) throws
    TelegramApiException{
       SendMessage message = new SendMessage();
       String[] tasks = {"Report about the
           victim's condition"};
       ReplyKeyboardMarkup replyKeyboardMarkup
           = makeKeyboard(tasks);
       message.setReplayMarkup(
           replyKeyboardMarkup);
       message.setText(mes);
       sendTo(message, chatId);
    }
```

## *5. Action language*

AL compiler has been developed with ANTLR compiler which enables to define a grammar in a ANTLR grammar language and compile it to the Java classes which represent the lexer and the parser of AL. The code generated by ANTLR parses the

AL code and apply specified semantical rules to the constructed syntax tree. To translate the target Java code while traversing the nodes of this tree, the syntax tree visitors were created that generate Java code from the initial AL code.

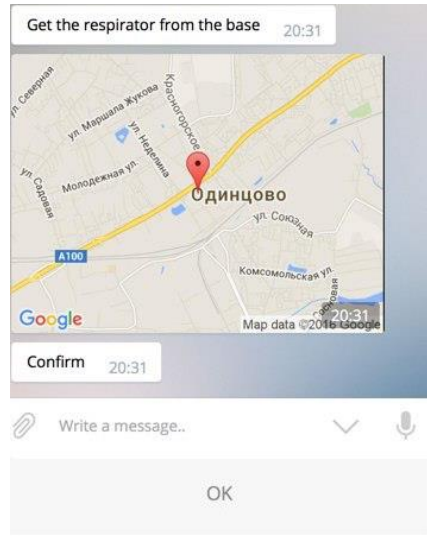


*Fig. 3: The agent is confirming the task implementation*


*Fig. 4: The rescuer agent Telegram mobile client interface.*

## 5.1 AL grammar

Here we provide the formal definition of suggested Action Language. The suggested Action Language is specific to Rescue and Search operations domain, and shall be reconsidered for other applications of the technology.

- < SystemNet> ::= < SP> : < name> — the name of system net
- < ElementNet> ::= < EP> : < name> — the name of element net
- < file> ::= file(< text>) – loads file from file-system
- < initialization > ::= < variable> = < value> — it is possible to assign variables of the types < file>, < float>, < string>
- < sendMessage> ::= sendMessage( < file> | < text> | < variable>) — sends a message from a transition of an element net
- < sendPhoto> ::= sendPhoto(< file> | < variable>) – send a Photo from a transition of an element net
- < sendLocation> ::= sendLocation(longitude : < variable> | < float>, latitude : < variable> | < float> — sends Location
- < sendVideo> ::= sendVideo(< file> | < variable>) – sends Video
- < sendAudio> ::= sendAudio(< file> | < variable>) – sends Audio
- < transition element net> :: = < name> = < text> response: (< sendAudio> | < sendVideo> | < sendLocation> | < sendPhoto> | < sendMessage>)* – this is the structure of the code which should be inscribed on the distinct transition of an element net.
- < connect> ::= connect (< name>.< transition>) – links a transition from an element net to a transition of a system net.
- < display> ::= display() – displays the object received on a transition of a system net
- < save> ::= save(< file> | < text> | < variable>) – saves the object received on the transition of a system net
- < transition system net> ::= < name> = (receive (photo | video | audio | message | location) : (save | display)*)*
- < loop> ::= forall < variable> in < variable botVariable.add(< variable>)

The AL example on Table 1 is from our search-and-rescue system model which was provided in the motivating example. It illustrates what kind of code must be inscribed to the transition of the Medical staff element net (Fig. 5) and the coordinator system net (Fig. 5). We will not provide the code for Rescue element net because it follows the same pattern of coding as for the Medical staff element net.

## *6. The application of the Telegram bot code generation technology*

In this section, we examine the application of the suggested technology to the motivating example provided in the section 3. The main components of the system are modelled with system net and element nets. Then the codegenerator translates NP-nets into Telegram bots components of the target Telegram-based multi-agent system being constructed.

The bot server serves the received requests according to the NP-net system net behaviour and sends the answers to the agents. All the actions, except the actions described on the system net transitions, of the developed Search and Rescue operation are handled by the Bot automatically. However, it is possible to interact ad-hoc during the operation, i.e. if an agent sends any kind of request that was not described by AL, the coordinator will be notified and will be able to answer this request with the standard Telegram client interface. All the event that were described with the system net transitions require the direct interaction of the coordinator. The agent will not be allowed to proceed to the next stage of operation, unless he receives the answer from the coordinator.

As soon as we launch the compiled bot, all the rescuers and medicals that were loaded to the system will receive notifications from the Telegram bot. The concurrent transitions (e.g. Helmet, Respirator, Gloves) from the Rescuer element net allow that all the actions inscribed on them could be executed by agents in any order. An agent will not be allowed to the next stage unless he performed all of them. After performing an action, the agent must confirm that in the mobile client by pressing the «OK» button (Fig. 3). The button appears on the screen when the agent has actions-transitions to fire.

When an agents reaches the “Begin the operation” action, the bot moves to the awaiting state and notifies the coordinator, that the agent has reached the state and waits till the next instructions will be provided. As soon as the coordinator fill the form and submit the answer, the agent will be allowed to move to the next state of his plan. That is due the «Begin the operation» transition is synchronized with the T2 system net transition.

## *7. Related works and further directions*

The codegeneration from models to executable software artifacts has attracted attention when model driven development became industrial popular and valuable approach [14]. The codegeneration from Petri net like models to executable software systems is studied for many formalisms and semi-formal industrial modelling languages like UML[15], [16] and SDL[17]. In [18], [19] the code generation tool for Input-Output Place-Transition Petri Nets was developed. In [20] the application of Sleptsov nets for modelling and implementation of hardware systems is studied. In [21], the technology to construct embedded access control systems from coloured Petri nets models is suggested. The approach to generate

C++ code from SDL models is developed in [8]. The code generation from the UML state machines[15] and sequence [16] diagrams to executable code was studied. These are a lot of studies in the field, so we only cited a few.

The translation from NP-nets to coloured Petri nets was developed in [8]. The translation from NP-nets to PROMELA models to verify the correctness of LTL properties is studied in [22]. The automatic translation from NP-nets models to distributed systems components that preserve liveness, conditional liveness, and safety properties was studied in [13]. In the current work, we adopted the translation scheme developed in the latter work to obtain executable code from the structure of NP-nets models.

The further research concerns theoretical as well as practical aspects of the developed automatic codegeneration system. From the theoretical point of view, it is interesting to study preservation of different behavioural properties by the implemented translation and securing different behavioural consistencies of generated systems and initial models. As the underlaying technologies are too large to conduct exhaustive formal verification, the both dynamic and static behavioural analyses techniques should be applied to study the correctness of the translation. From the practical point of view, there are lot of attractive features that are to be implemented. For example, it is not possible to change the deployed bots at runtime in the tool. However, such function could be of use for long term operations, when new actions should be integrated into an operating Telegram system without recompiling the whole system. The runtime deployment will be considered in the future research. Also, the scalability of generated Telegram systems and possible schemes of agents distribution in the system are the subjects of the further research.

## *8. Conclusion*

The developed technology enables developers to create Telegram Bots according to a visually clear model that could be verified and tested with help of the developed methods [22], [8], [9]. It allows to create distributed event-based Telegram Bots systems that operate on the Telegram platform and the AL language supports all the features provided by Telegram Bot API up to the moment.

The automatic code-generation reduces the risk of introducing defects on the implementation phase of software development process and improves the quality of the resultant code. It not only reduces the cost of software production, but also makes the quality of developed systems more predictable. The suggested technology is demonstrated with the example of a Search and Rescue system.

The authors would like to thank the anonymous referees for valuable and helpful comments.

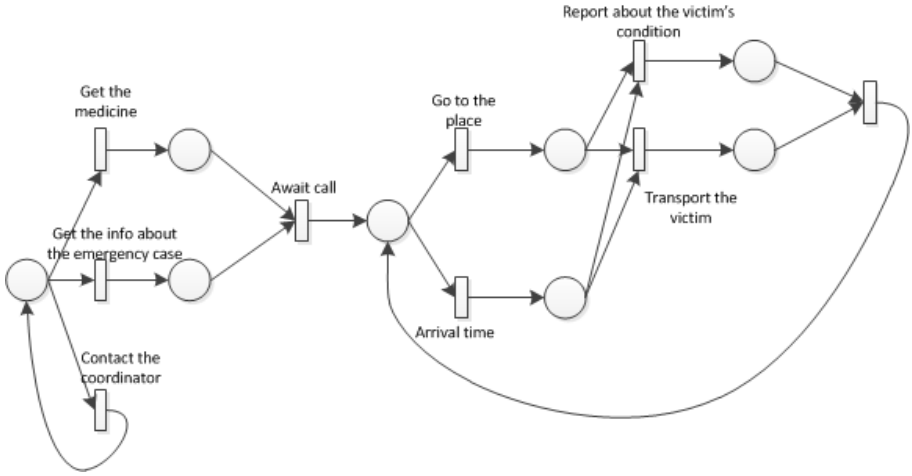


*Fig. 5. The Medical Staff element net*

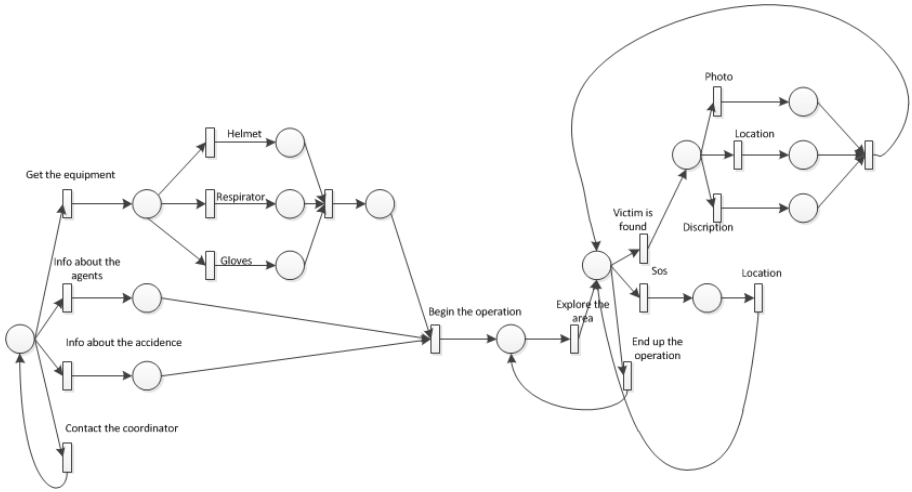


*Fig. 6. The Rescuer element net*

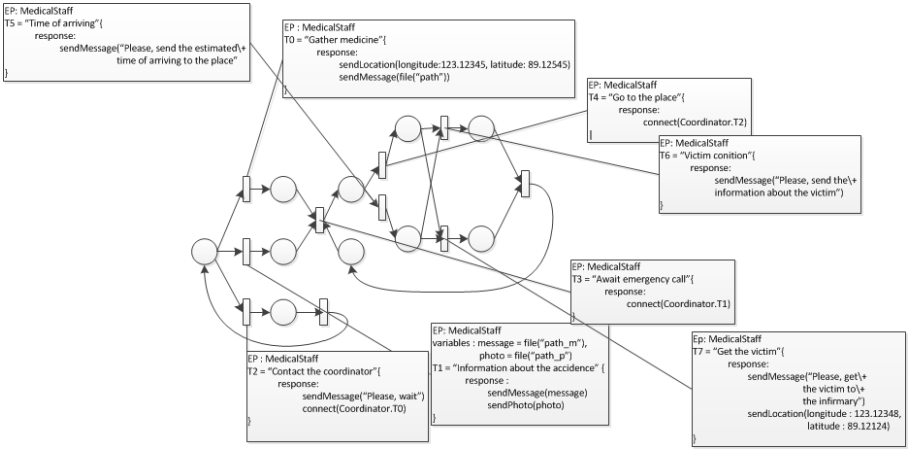


*Fig. 7. The element net augmented with code*

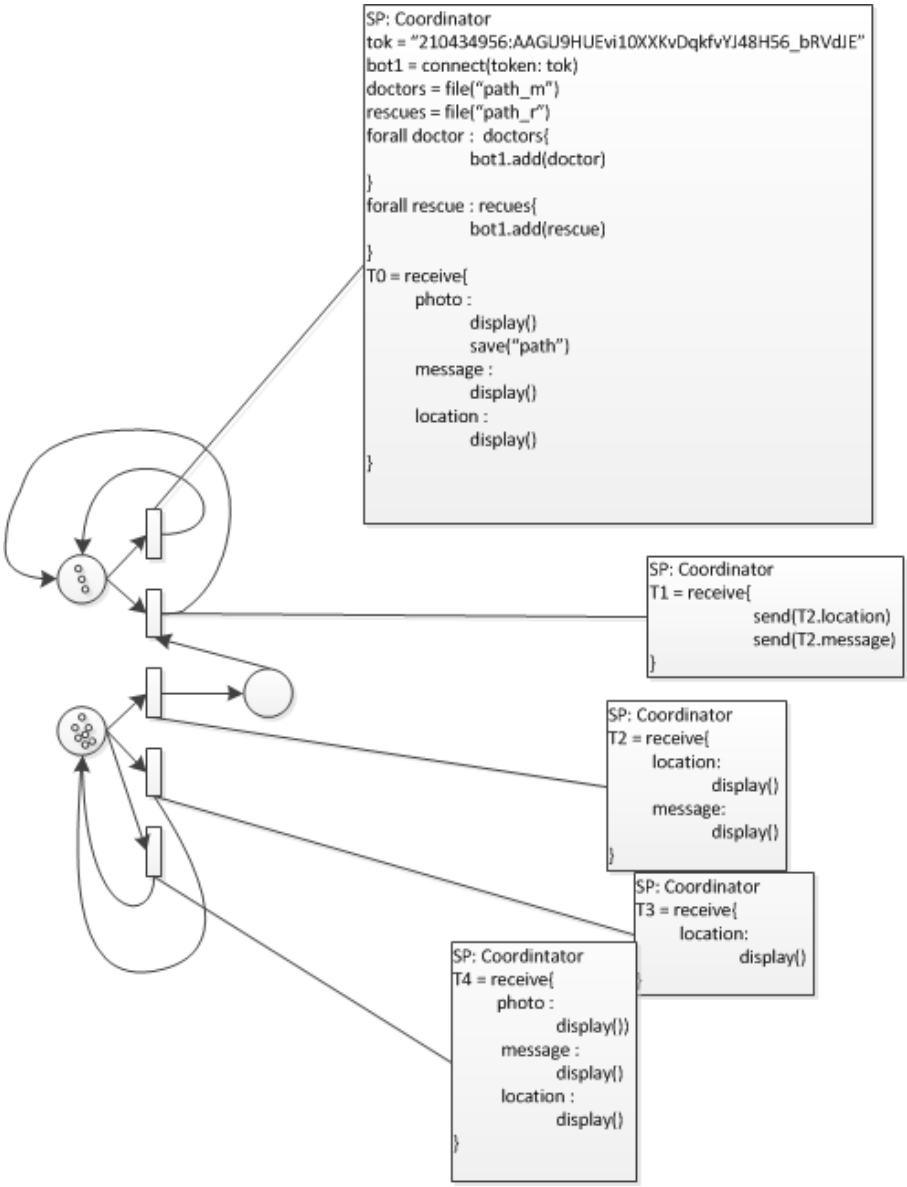


*Fig. 8. The system net augmented with code*

## Acknowledgement

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16-01-00546.

## References

[1]. Telegram Bot API online documentation. [Online]. Available at: https://core.telegram.org/bots/api
[2]. L. Chang, X. He, J. Li, and S. M. Shatz. Applying a Nested Petri Net Modelling Paradigm to Coordination of Sensor Networks with mobile agents. In Proc. of Workshop on Petri Nets and Distributed Systems. Xian, China, 2008, pp. 132–145.
[3]. I. A. Lomazova, “Nested Petri Nets - a Formalism for Specification and Verification of Multi-Agent Distributed Systems,”Fundamenta Informaticae, vol. 43, no. 1, pp. 195–214, 2000.
[4]. Nested Petri nets: Multi-level and Recursive Systems. Fundamenta Informaticae, vol. 47, no. 3-4, pp. 283–293, Oct 2001.
[5]. Nested Petri Nets for Adaptive Process Modelling. In Pillars of Computer Science, ser. Lecture Notes in Computer Science, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Springer Berlin Heidelberg, 2008, vol. 4800, pp. 460–474.
[6]. K. Hoffmann, H. Ehrig, and T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In ICATPN, 2005, pp. 268–288.
[7]. D. Frumin and L. Dworzanski. NPNtool: Modelling and Analysis Toolset for Nested Petri Nets. In Proceedings of the 7th Spring/Summer Young Researchers Colloquium on Software Engineering, 2013, pp. 9–14.
[8]. L. Dworzanski and I. A. Lomazova. CPN Tools-Assisted Simulation and Verification of Nested Petri Nets. Automatic Control and Computer Sciences, vol. 47, no. 7, pp. 393–402, 2013.
[9]. L. Dworzanski and I. A. Lomazova. On Compositionality of Boundedness and Liveness for Nested Petri Nets. Fundamenta Informaticae, vol. 120, no. 3-4, pp. 275–293, 2012.
[10]. The Ministry of the Russian Federation for Civil Defence. Emergencies and Elimination of Consequences of Natural Disasters. Emergency Cases Registered in Russia. [Online]. Available at: http://25.mchs.gov.ru/document/2644168
[11]. (2013) United States Coast Guard Search and Rescue Summary Statistics. [Online]. Available at: https://www.uscg.mil/hq/cg5/cg534/SARfactsInfo/SAR%20Sum%20Stats%2064-13.pdf
[12]. T. Parr. The Definitive ANTLR 4 Reference. 2nd ed. Pragmatic Bookshelf, 2013.
[13]. L. Dworzanski and I. A. Lomazova. Automatic Construction of Distributed Component System from Nested Petri Nets. In print, Programmirovanie, vol.6, 2016 (in Russian).
[14]. B. Selic. The Pragmatics of Model-Driven Development. IEEE Software, vol. 20, no. 5, p. 19, 2003.
[15]. A. Knapp and S. Merz. Model Checking and Code Generation for UML State Machines and Collaborations. Proc. 5th Wsh. Tools for System Design and Verification, pp. 59–64, 2002.
[16]. D. Kundu, D. Samanta, and R. Mall. Automatic Code Generation From Unified Modelling Language Sequence Diagrams. Software, IET, vol. 7,no. 1, pp. 12–28, 2013.

[17]. P. Morozkin, I. Lavrovskaya, V. Olenev, and K. Nedovodeev. Integration of SDL Models into a SystemC Project for Network Simulation. In SDL 2013: Model-Driven Dependability Engineering: 16th International SDL Forum, Montreal, Canada, June 26-28, 2013. Proceedings. Springer Berlin Heidelberg, 2013, pp. 275–290.

[18]. L. Gomes, J. P. Barros, A. Costa, and R. Nunes. The Input-Output Place-Transition Petri Net Class and Associated Tools. In Industrial Informatics, 2007 5th IEEE International Conference on, vol. 1. IEEE, 2007, pp. 509–514.

[19]. R. Campos-Rebelo, F. Pereira, F. Moutinho, and L. Gomes. From IOPT Petri Nets to C: An Automatic Code Generator Tool. In Industrial Informatics (INDIN), 2011 9th IEEE International Conference on.

[20]. D. Zaitsev and J. Jurjens. Programming in the Sleptsov Net Language For Systems Control. Advances in Mechanical Engineering, vol. 8, no. 4, p. 1-11, 2016. DOI: 10.1177/1687814016640159.

[21]. K. H. Mortensen. Automatic Code Generation Method Based on Coloured Petri Net Models Applied on an Access Control System. In Application and Theory of Petri Nets 2000. Springer, 2000, pp. 367–386.

[22]. M. L. F. Venero and F. S. C. da Silva. Model Checking Multi-Level and Recursive Nets. Software & Systems Modeling, pp. 1–28, 2016.

# Автоматическая генерация кода по вложенным сетям Петри для систем на основе событий на платформе Telegram

*Д.И Самохвалов<disamokhvalov@edu.hse.ru>*
*Л.В. Дворянский <leo@mathtech.ru>*
*Национальный исследовательский университет Высшая школа экономики,*
*ул. Мясницкая., 20, Москва, 101000, РФ.*

**Аннотация.** Вложенные сети Петри – это расширение формализма раскрашенных сетей Петри, которые используют сети Петри в качестве фишек. Данный формализм позволяет создавать подробные модели мультиагентных систем, осуществлять имитационное моделирование, верифицировать и анализировать их свойства на формальном и строгом уровне. Мультиагентные системы находят применение во многих областях – начиная системами, для которых безопасность играет критическую роль, заканчивая повседневными системами, работающими на персональных вычислительных устройствах. Число таких систем в современном мире растет вместе с увеличивающимся числом мобильных вычислительных устройств. На данный момент разработаны инструменты и методы моделирования и анализа вложенных сетей Петри, но синтез мультиагентных систем по моделям вложенных сетей Петри еще недостаточно исследован и находится в стадии активного изучения. Метод автоматической генерация исполняемого кода целевой системы по спроектированной и верифицированной модели вложенной сети Петри обеспечивает получение корректных системы из корректных спецификаций на языке вложенных сетей Петри. В данной работе, демонстрируется применение формализма вложенных сетей Петри для построения модели системы управления поисковыми и спасательными операциями и

автоматической генерации реализации в виде исполняемого кода событийно-управляемых систем основанных на платформе Telegram. Мы добавляем возможность аннотировать модели вложенных сетей Петри с помощью Action Language, который позволяет связывать срабатывания переходов на модельном уровне с вызовами Telegram Bot API на уровне реализации. Предложенный подход продемонстрирован на примере аннотированной модели системы координирования спасательной операции.

**Ключевые слова:** вложенные сети Петри;, telegram bot api; язык действий; событийно-управляемые системы; кодогенерация.

**DOI:** 10.15514/ISPRAS-2016-28(3)-5

Для цитирования: Самохвалов Д.И., Дворянский Л.В. Автоматическая генерация кода по вложенным сетям Петри для систем на основе событий на платформе Telegram. *Труды ИСП РАН,* том 28, вып. 3, 2016. стр. 65-84 (на английском). DOI: 10.15514/ISPRAS-2016-28(3)-5

## Список литературы

[1]. Telegram Bot API online documentation. [Online]. Доступно по ссылке: https://core.telegram.org/bots/api
[2]. L. Chang, X. He, J. Li, and S. M. Shatz. Applying a Nested Petri Net Modelling Paradigm to Coordination of Sensor Networks with mobile agents. In Proc. of Workshop on Petri Nets and Distributed Systems. Xian, China, 2008, pp. 132–145.
[3]. I. A. Lomazova, “Nested Petri Nets - a Formalism for Specification and Verification of Multi-Agent Distributed Systems,”Fundamenta Informaticae, vol. 43, no. 1, pp. 195–214, 2000.
[4]. Nested Petri nets: Multi-level and Recursive Systems. Fundamenta Informaticae, vol. 47, no. 3-4, pp. 283–293, Oct 2001.
[5]. Nested Petri Nets for Adaptive Process Modelling. In Pillars of Computer Science, ser. Lecture Notes in Computer Science, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Springer Berlin Heidelberg, 2008, vol. 4800, pp. 460–474.
[6]. K. Hoffmann, H. Ehrig, and T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In ICATPN, 2005, pp. 268–288.
[7]. D. Frumin and L. Dworzanski. NPNtool: Modelling and Analysis Toolset for Nested Petri Nets. In Proceedings of the 7th Spring/Summer Young Researchers Colloquium on Software Engineering, 2013, pp. 9–14.
[8]. L. Dworzanski and I. A. Lomazova. CPN Tools-Assisted Simulation and Verification of Nested Petri Nets. Automatic Control and Computer Sciences, vol. 47, no. 7, pp. 393–402, 2013.
[9]. On Compositionality of Boundedness and Liveness for Nested Petri Nets. Fundamenta Informaticae, vol. 120, no. 3-4, pp. 275–293, 2012.
[10]. The Ministry of the Russian Federation for Civil Defence. Emergencies and Elimination of Consequences of Natural Disasters. Emergency Cases Registered in Russia. [Online]. Доступно по ссылке: http://25.mchs.gov.ru/document/2644168
[11]. (2013) United States Coast Guard Search and Rescue Summary Statistics. [Online]. Доступно по ссылке:: https://www.uscg.mil/hq/cg5/cg534/SARfactsInfo/SAR%20Sum%20Stats%2064-13.pdf

[12]. T. Parr. The Definitive ANTLR 4 Reference. 2nd ed. Pragmatic Bookshelf, 2013.
[13]. Дворянский Л.В., Ломазова И.А. Автоматическое построение распределенной компонентной системы по вложенной сети Петри. В печати: Программирование, no. 6, 2016 (in Russian).
[14]. B. Selic. The Pragmatics of Model-Driven Development. IEEE Software, vol. 20, no. 5, p. 19, 2003.
[15]. A. Knapp and S. Merz. Model Checking and Code Generation for UML State Machines and Collaborations. Proc. 5th Wsh. Tools for System Design and Verification, pp. 59–64, 2002.
[16]. D. Kundu, D. Samanta, and R. Mall. Automatic Code Generation From Unified Modelling Language Sequence Diagrams. Software, IET, vol. 7,no. 1, pp. 12–28, 2013.
[17]. P. Morozkin, I. Lavrovskaya, V. Olenev, and K. Nedovodeev. Integration of SDL Models into a SystemC Project for Network Simulation. In SDL 2013: Model-Driven Dependability Engineering: 16th International SDL Forum, Montreal, Canada, June 26-28, 2013. Proceedings. Springer Berlin Heidelberg, 2013, pp. 275–290.
[18]. L. Gomes, J. P. Barros, A. Costa, and R. Nunes. The Input-Output Place-Transition Petri Net Class and Associated Tools. In Industrial Informatics, 2007 5th IEEE International Conference on, vol. 1. IEEE, 2007, pp. 509–514.
[19]. R. Campos-Rebelo, F. Pereira, F. Moutinho, and L. Gomes. From IOPT Petri Nets to C: An Automatic Code Generator Tool. In Industrial Informatics (INDIN), 2011 9th IEEE International Conference on.
[20]. D. Zaitsev and J. Jurjens. Programming in the Sleptsov Net Language For Systems Control. Advances in Mechanical Engineering, vol. 8, no. 4, p. 1-11, 2016. DOI: 10.1177/1687814016640159.
[21]. K. H. Mortensen. Automatic Code Generation Method Based on Coloured Petri Net Models Applied on an Access Control System. In Application and Theory of Petri Nets 2000. Springer, 2000, pp. 367–386.
[22]. M. L. F. Venero and F. S. C. da Silva. Model Checking Multi-Level and Recursive Nets. Software & Systems Modeling, pp. 1–28, 2016.