Software System Behavior Can Be Analyzed with Visual Analytics

Yaroslav V. Kotylev[a], Alexey A. Mitsyuk[a]

[a]National Research University Higher School of Economics,
20 Myasnitskaya St., Moscow 101000, Russia

Abstract

Enterprise software systems are usually large and complex. They can have complicated architecture and millions of different states. It is difficult to imagine a process of designing new software system or analysis of developed system without various modeling techniques. Visual presentation of the software system structure and behavior provides possibilities which can facilitate better analysis. Graphical notation of the Petri net formalism is often used for software system description. An extension of Petri net is the high-level Petri net formalism can be successfully used for the description of modular software systems. There are a lot of related tools that can be used for "static" visualization of high-level Petri net. The current work considers possible methods of dynamic visualization of high-level Petri nets which describe software system from behavioral perspective.

Keywords

Software Visualization, High-level Petri Nets, Dynamic Visualization

1. Introduction

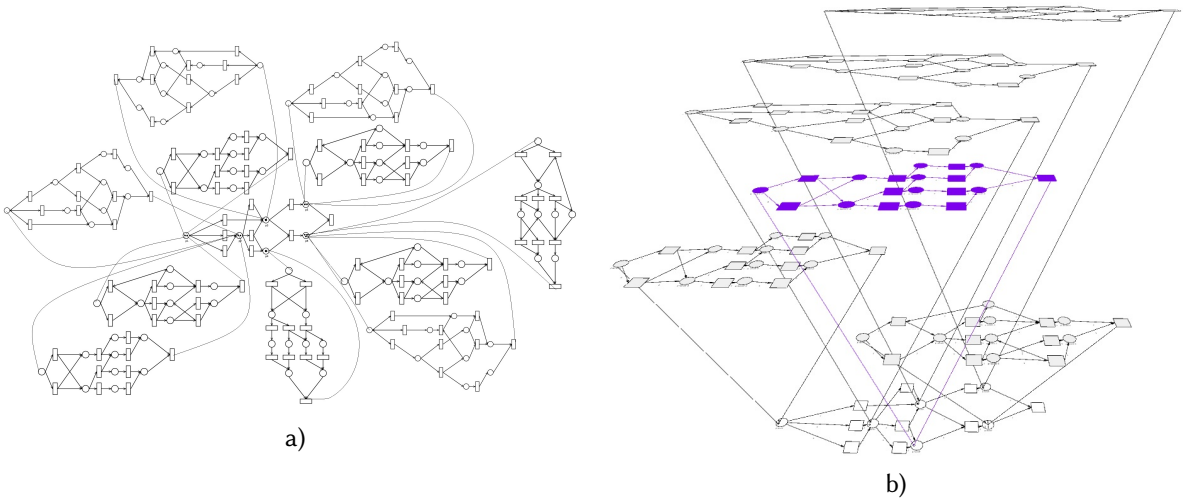
Complexity of enterprise software systems steadily grows year by year, release by release [1]. The software system modeling process has already become too difficult to design and analyze it without special models and tools. A Petri net is a mathematical modeling language for the description of distributed systems. A Petri net has not only precise mathematical notation, but a graphical notation as well [2]. These two aspects made Petri nets useful for several areas such as: process modeling, software system design and analysis, data analysis, etc.

Modern software systems are characterized by large number of independent processes. For instance, they are often implemented in service-oriented style. Service-oriented architecture (SOA) represents a system as a set of independent components which provide services to another modules via communication protocols over a network. Components often have shared resources, for example, common data storage. As a result, enterprise systems confront two important issues: data integration and data sharing [3]. How to prove or at least analyze correctness of such systems? perception is based on graphical visualization, Petri nets could be used for those purposes.

Modern enterprise systems are too complex to analyze them as a single artefact. So that displaying software systems as a plain graph will not be helpful. There are several approaches to reduce the perception of complexity such as grouping or abstraction. For example, a dozen of modules of the

Modeling and Analysis of Complex Systems and Processes - MACSPro'2020, October 22–24, 2020, Venice, Italy & Moscow, Russia

yavkotylev@edu.hse.ru (Y.V. Kotylev); amitsyuk@hse.ru (A.A. Mitsyuk)
https://pais.hse.ru/en/ (A.A. Mitsyuk)
0000-0003-3290-1543 (Y.V. Kotylev); 0000-0003-2352-3384 (A.A. Mitsyuk)



a) b)

Figure 1: Plane manual-made visualization of a nested Petri net a) and its layered visualization in 3D b) [5]

system could be grouped by any common feature and could be considered as a single element with hidden details.

Hierarchical high-level Petri net is an extended formalism with the basic idea of constructing a large model by combining a number of small Petri nets into a larger net [4]. A hierarchical high-level Petri net often can be translated into a behaviorally equivalent ordinary Petri net. In case of enterprise system structure visualization, each part of the system could be depicted as an ordinary Petri net at the lowest level of visualization abstraction and the set of modules could be drawn as a single element at the highest level of abstraction.

But visualization of "static" structure of the system does not cover all aspects of systems description. Behavioral (or "dynamic') aspect of a system is as important as its structure. In this paper we try to find ways to answer the question "How to visualize dynamics of a complex hierarchical system?"

2. Software system analysis with high-level Petri nets visualization

As Petri nets language is not only a mathematical formalism but graphical notation as well, it could be used for software system visualization. We have already mentioned in Introduction section that modern enterprise software systems are developed with a module-based approach. Modules can be fully separated on the same level of hierarchy or nested into other modules, and anyway their internal states are independent and communication between modules carried out overall network. But nonetheless the modules often share common resources such as data. The extension of ordinary Petri nets is the high-level Petri net formalism. It could be used for building visualizations of such systems. Each module of the system can be represented as a single ordinary Petri net. And then, a set of ordinary Petri nets can be aggregated to a hierarchical high-level Petri net to represent system as a whole.

There are several approaches for high-level Petri net visualization. One of them is hierarchical layered-layout visualization [5]. The main idea of layered-layout approach is to display each element net onto a single layer on distinct level with links to main net. Figure 1 a) shows a manual-made visualization of a nested Petri net [6], and Figure 1 b) shows the same nested Petri net visualized using layered layout (some element nets are collapsed).

This method is useful for static analysis of Petri net structure, when users are able to collapse needless element nets to hide redundant structures and expand element nets which are needed to be analyzed to investigate them in detail. But this approach does not support any dynamic features. In this work, we are going to consider dynamic visualization of real behavior of the system.

2.1. What does it mean "real behavior" and how to analyze it?

Let's imagine that user has already had a software system prototype or a fully developed system. And the model of the system is described in the high-level Petri net formalism. It is possible to run the system on a test data and gather produced logs. Surely the logging part of the system should be prepared in advance: logs should be presented entirely and in appropriate format. Each event record should contain at least:

- timestamp;
- identifier of log activity (it's needed to define a related transition of the Petri net model);
- thread identifier (for multi-thread or multi-agent environments).

These logs can be used for replaying and visualizing system behavior on a high-level Petri net model.

2.2. Existing approaches to visualize process dynamics in information systems

In this project, our aim is to achieve convenient visualization techniques by combining ideas from graph drawing, process visualization, and introducing completely new ideas.

Existing approaches to visualize process dynamics in information systems came mostly from process modeling and process mining fields. In process modeling, many tools have been developed to edit process models, show them graphically. They allow us to view and change model structure. Such tools often enhanced with a simulation module/plug-in that is able to execute model step-by-step showing changes in model state to the user.

Let us consider several examples of modeling tools and editors. One of the tools applied in business process management education is WoPeD [7] (https://woped.dhbw-karlsruhe.de/). It is suitable for working with static workflow nets, editing their structure, calculating properties of these models.

Another tool to deal with ordinary Petri nets is called Carassius [8] (https://pais.hse.ru/research/projects/carassius). The tool is typical for such type of applications. It supports editing and visualizing of Petri nets shown on a plane view.

Carassius supports simulation of ordinary Petri nets. In this mode, a user can study how a system modeled by the Petri net is executed due to P/T-net semantics. Figure 2 shows how Petri net simulation looks in Carassius. Here the intensity of blue coloring depicts the time from the firing of a transition. Yellow token is the one that has been produced at the last simulation step.

At the same time, approaches have been proposed to visualize static structure of high-level Petri nets of different types. Coloured Petri nets are one of the most commonly used type of high-level Petri nets. They support a concept of hierarchy and complex control-flow constructs. CPN Tools [9] (http://cpntools.org/) is a tool for Coloured Petri nets. Figure 3 shows the very basic CPN model.

Reference nets is another type of high-level Petri nets which support the concept of hierarchy. Also these nets support data-flow modeling. Renew [10] (http://www.renew.de/) is a well-designed tool for modeling, analyzing, and simulating reference nets. A basic sample model during simulation process is shown in Figure 4.

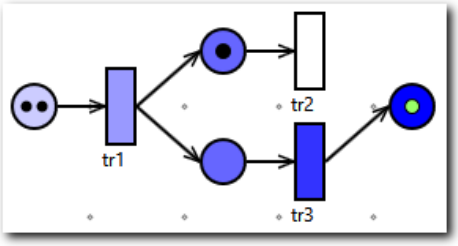


Figure 2: Petri net simulation in Carassius

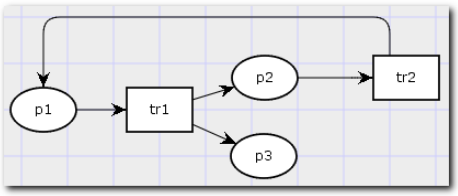


Figure 3: Coloured Petri net in CPN Tools

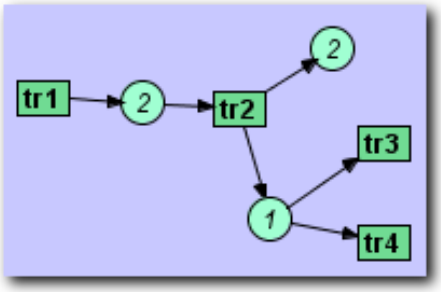


Figure 4: Reference net simulation in Renew

What is common in all these tools supporting Petri net modeling? Models are shown using basic graph drawing algorithms. Moreover, special algorithms for Petri net drawing can apply the fact that an ordinary Petri net is s bipartite graphs. Petri net structure can be shown in a flat view independently of their type. They have nodes and arcs of different types which are shown by rectangular, ellipse, and arrow forms of different appearance. Petri net marking is usually shown by tokens in place nodes. These tokens graphically depicted as dots or small circles inside larger places. Marking can also be shown by a number of tokens in a corresponding place (like in Figure 4).

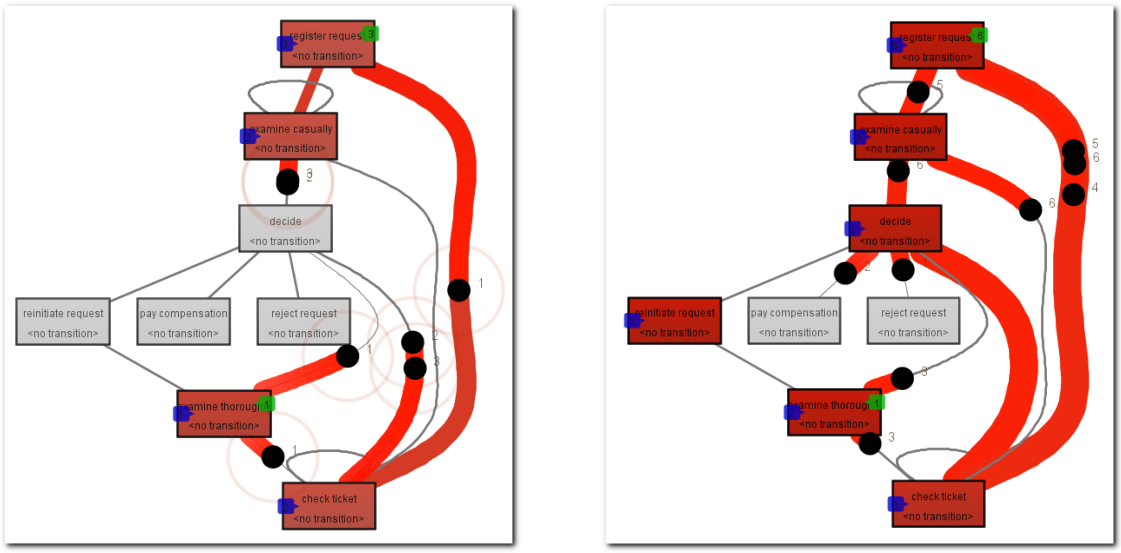
It is important to mention that usually editors can not really show dynamics. They show static pictures or GUI with structure and state of modeled system. These pictures are able to change because of the user action.

Rather different approach for visualizing process dynamics can be found in process mining. ProM Framework [11] is the most developed open source process mining tool. This tool supports different types of process visualization including dynamic views.

Fuzzy mining [12] is used to discover process models which show how process activities relate to each other via a directly follows relation and other relations. This approach is implemented as a ProM-plugin *Fuzzy miner*. This plug-in is equipped with an ability to show dynamic visualizations out of a model and an event log. Figure 5 shows how this visualization looks like.

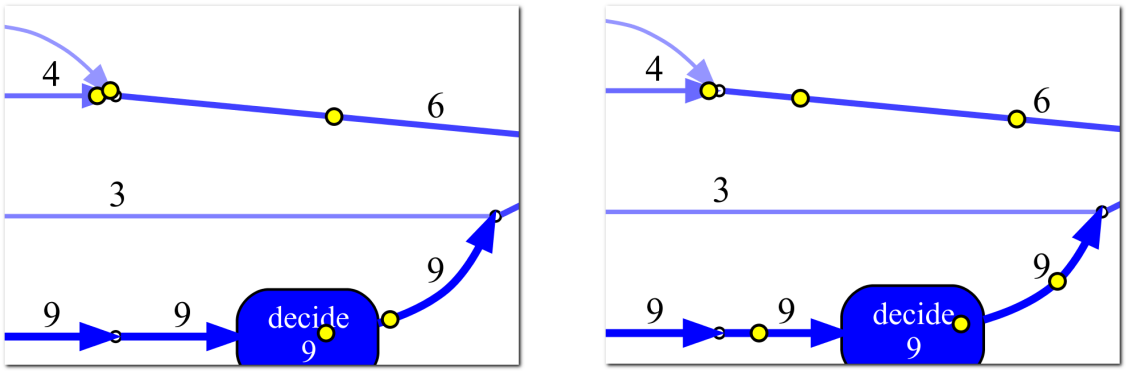
As earlier, we have a graph-based process model. Nodes represent process activities, while arcs represent relations between these activities. For example, an arc connects two nodes if there is a directly-follows relation between the two corresponding activities. Thus, the process structure is shown. But we also can see black tokens which are actually *moving* through the model. *Active* arcs are shown in red coloring. We can also see frequency of activities: frequent ones became dark-red after several times.

Another tool with the same capability is called Inductive visual miner [13]. It is also implemented



a) b)

Figure 5: Two moments (a and b) of a fuzzy process movie



a) b)

Figure 6: Two moments (a and b) of visualization supported by Inductive visual miner

as a ProM plug-in but now also exists as a separate tool. Inductive mining is one of the most popular process discovery algorithms as it allows us to discover a well-structured process model consuming a relatively short time period. Inductive miner is enhanced with a convenient GUI which is able to visualize dynamics of the discovered process as it is shown in Figure 6. Yellow tokens run through the model. More frequently executed parts of the model became more bluish with time.

Both these approaches show to the user how real behavior from and event log can be *replayed* on the corresponding process model. Commercial tools like Fluxicon Disco [14] (https://fluxicon.com/disco/) or Celonis (https://www.celonis.com/) support the same functionality.

Visual representation of process dynamics as *process movies* has been also researched for declarative on-line discovery [15]. Of interest is the approach of mapping event logs on specially prepared visual backgrounds to show process characteristics [16].

The goal of our project is to combine ideas of static graph-based model visualization for high-level Petri nets with the concept of "process movies" from process mining which allow us to show process

dynamics recorded in event logs.

2.3. Why you can not visualize high-level Petri net dynamics easily?

There are several difficulties related to dynamic visualization of software system behavior with high-level Petri nets. If the system is displayed as a whole, users will disperse concentration due to high amount of redundant information. On the other hand, if the most system modules (element nets) are collapsed as it feasibly in layered-layout approach, users will have a chance to lose important details hidden in collapsed nets. Therefore, a method of auto hiding/exposition of element nets should be proposed to maintain completeness-clarity visualization balance between details and system complexity.

2.4. Objectives

Let us summarize the main objective of the project. The main objective is to develop visual analytics methods which will successfully catch process dynamics in multi-agent systems represented as high-level Petri nets. Proposed methods should take into attention key properties of multi-agent systems: their complex and hierarchical multilevel structure; dynamic asynchronous agent activities, communications and collaborations; large number of events and activities in such distributed systems. It is of most importance to support user with visual tools to highlight the most important fraction of system behaviour while shading immaterial detail. As navigation device shows only roads which are important for driver, visual analytics tool should assist a researcher by highlighting the most significant model parts.

3. Completeness-clarity balance

We are going to introduce the *four variables* intended for automatic decision making on the displaying/hiding element Petri nets during dynamic visualization of high-level Petri net model of the system based on logs:

- rate of event importance;
- time interval of calculation;
- distance between time intervals;
- displaying threshold.

Let us consider proposed values on the example of a typical learning management information system (LMS). The system has a lot of different modules. Also, this system has been modeled using the high-level Petri net formalism. In graphical notation, each module of the system is depicted as an element Petri net. Also, system logs were translated into Petri net run. For example, the system has an event - user submitted login and password. This event could be transformed into the following part of the run: input tokens t_1 (login), t_2 (password) were removed from the place Pl_1 and output token t_3 was added to the Pl_2 with firing of transition T_n.

One of the modules is an academic discipline support module. This module provides functions related to learning process such as updating course info, uploading homework exercises by students.

3.1. Rate of event importance

The idea is to assign weights or measure of importance for each activity of the system. This measure should be allocated by user to distinguish significant events and secondary events. The value of *rate of event importance* will be taken into account to make a decision whether to display the element net to which the activity belongs or not. The value of *rate of event importance* belongs to the interval $[0; +\infty)$.

For instance, the system has logs for two types of activities: *visiting "discipline info" page* by any user, and *changing a course material*. We can assume that the activity of changing course materials is much more important than the activity of visiting info-page by a single user. It doesn't make sense to display the element net of the discipline each time when users visit the info-page. At the same time we expect that the element net will be displayed at the moment of changing course materials because it may lead to additional activities such as user notification, database updates, uploading new files, etc. Thus, based on a subjective opinion we can assign "0" *rate of event importance* to info-page visit event and "20" to material course update event.

3.2. Time interval of calculation

In order to analyze dynamic behavior of the system, we propose to split up logs of each module by time intervals and make a decision of displaying an element net for each interval separately. Let's examine new activity of the academic discipline support module of the LMS — *uploading homework exercise*s by students. Event of this type are a meaningful activity with *rate of event importance* greater than zero. We can assume that in terms of behavioral analysis, we don't care about infrequent homework loadings. Contrariwise, we definitely know that frequency of uploads rises sharply before a deadline. The deadline may lead to updating course materials (new home tasks, for example), changing discipline description or even to the finish of the academic discipline. *Time interval of calculation* provides the ability to measure the frequency of process activities (events). On the subject of the previous example, we would like to display element net of the module before a deadline, when rate of posting homework by students is bigger than 100 per hour. "Hour" is a *time interval of calculation*. The value of *time intervals of calculation* could be tuned for each element Petri net of high-level Petri net individually.

3.3. Time gap between intervals

Time gap between intervals — time between beginnings of two nearest *time intervals of calculation*. The value of *time gap between intervals* should not be greater than *time interval of calculation* and could be tuned for each element Petri net of high-level Petri net individually as well as *time intervals of calculation*.

3.4. Displaying threshold

Displaying threshold is the final value for decision making about displaying the certain element net for a chosen time interval. It should be compared with *level of interval importance* (see Equation 1) — sum of events from the chosen time interval multiplied by events' *rate event of importance*. If the result of calculation is greater than the value of *displaying threshold*, then the element net should be drawn fully. If the result of calculation is smaller than the value of *displaying threshold*, then the element net should be hidden for a time interval. The value of *threshold of displaying* could be unique for each element net.

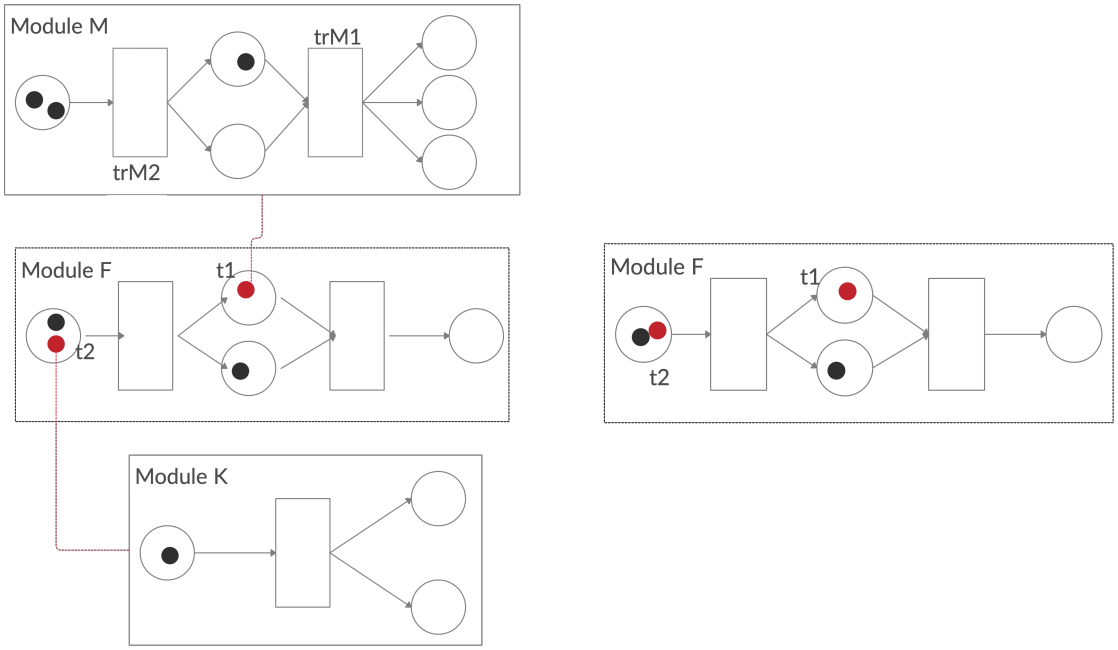


Figure 7: Schematic representation of the software system with high-level Petri net

$$Level\ of\ interval\ importance:\ \ LoII_i = \sum_{k=1}^{f} n_{ik} * r_k \tag{1}$$

where:

i = time interval identifier,
k = type of chosen event,
n_{ik} = number of k-type events within the i interval,
r_k = *rate event of importance* of k-type event.

3.5. How can we apply the proposed completeness-clarity balance method?

Let's consider a simple example. There is a pack of logs for a software system with three different modules — F, M and K). Modules M and K are submodules of the module F, and they are "packed" in tokens t_1 and t_2 of the module F. This system has the high-level Petri net representation. Figure 7 shows the system with all modules at the a) part and shows the system with hidden submodules at the b) part.

Module M has three types of events: a_1, a_2 and a_3. Event a_1 is very important activity and we don't want to lose visualization of the element net of the the M module when a_1 happens. This event relates to transition $trM1$. Event a_2 (happens when the transition $trM2$ is firing) doesn't deserve attention as a unique activity, but it will be useful to analyze system behavior in case of sequence of a_2 events. And finally, a_3 event is not significant at all.

The following values were chosen as parameters for completeness-clarity balance method:

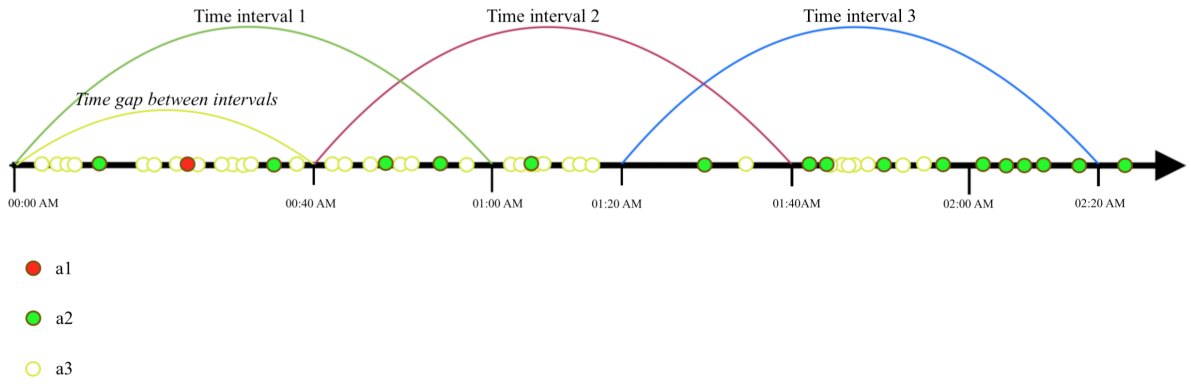


Figure 8: Completeness-clarity balance example

- rate of event importance:
  - for a_1 = 90,
  - for a_2 = 10,
  - for a_3 = 0;
- time interval of calculation = 1 hour;
- distance between time intervals = 40 minutes;
- displaying threshold = 90.

Figure 8 shows the example of a_1, a_2 and a_3 distribution through the time period from 00:00 AM to 02:20 AM. There are three intertwined time intervals (equals to *time interval of calculation* = 1 hour) with distance equals to *time gap between intervals* = 40 minutes:

- time interval 1 - from 00:00AM to 01:00AM contains 1 a_1, 4 a_2 and 19 a_3;
- time interval 2 - from 00:40AM to 01:40AM contains 0 a_1, 4 a_2 and 13 a_3;
- time interval 3 - from 01:20AM to 02:40AM contains 0 a_1, 10 a_2 and 8 a_3;

Using Equation 1 we can calculate *level of interval importance (LoII)* for each interval:

- $LoII_{time\ interval\ 1} = 1 * 90 + 4 * 10 + 19 * 0 = 130$;
- $LoII_{time\ interval\ 2} = 0 * 90 + 4 * 10 + 13 * 0 = 40$;
- $LoII_{time\ interval\ 3} = 0 * 90 + 10 * 10 + 8 * 0 = 100$.

And finally we can compare each LoII with *displaying threshold (DT)* = 90. The results of calculation show that element net will be displayed for $time\ interval\ 1$, $time\ interval\ 3$ and will be hidden for $time\ interval\ 2$:

Time interval	LoII of interval	LoII > DT	Element net will be displayed
Time interval 1	130	Yes	Yes
Time interval 2	40	No	No
Time interval 3	100	Yes	Yes

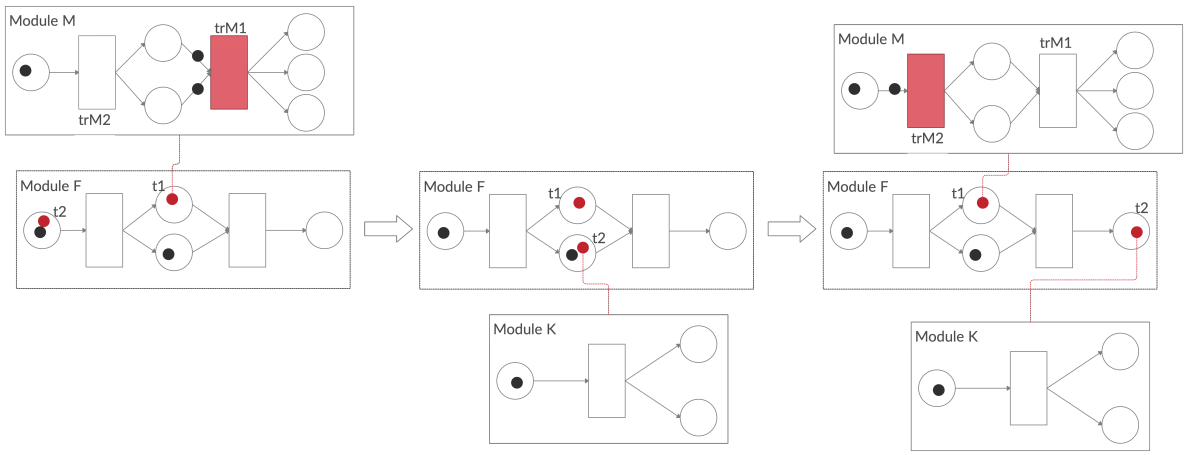


Figure 9: The system states by time intervals

Figure 9 shows three states of the system: the left state at the *time interval* 1 (we can see the a_1 event - $trM1$ is firing), the central state during the *time interval* 2 (the module M is hidden) and the right one is about *time interval* 3, when the element net for the module M appears again due to a_2 event. Please note that for the example module K has its own events and the time when the element net is displayed differs from the module M.

4. Conclusions and Future Work

In this paper we presented the first ideas on how process dynamics can be visualized based on hierarchical high-level Petri nets. The paper documents a first part of development of dynamic visualization tool of high-level Petri net. The result of the research will provide methods of dynamic Petri net visualization and make it possible to develop visualization tool.

The visualization tool will be executed together with simulation/log processing engine. This engine will be a back-end part of the product which will scan real software systems' logs and send related commands to visualization tool. Working together log processing engine and visualization tool will provide the user with the ability to analyze structure and behavior of chosen software system.

Dynamic visualization based on real logs can help to find bottlenecks and unwanted processes interaction. Also, it will help to analyze interaction of different modules with shared resources. A hierarchical approach of the software system visualization will allow user to get rid of redundant details by wrapping layers of hierarchy. It means that dynamic visualization of the software system via high-level Petri net will combine advantages of static (structural) and dynamic (behavioral) visualization for more convenient analysis of the software systems.

Acknowledgments

This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

References

[1] M. M. Lehman, Programs, life cycles, and laws of software evolution, Proceedings of the IEEE 68 (1980) 1060–1076. doi:10.1109/PROC.1980.11805.

[2] J. Desel, G. Juhás, "what is a petri net?", in: Unifying Petri Nets, volume 2128 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 1–25.

[3] P. Ai, Z. Wang, Y. Mao, Service-oriented architecture of specific domain data center, in: GCC, volume 3251 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 855–858.

[4] K. Jensen, Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 1, Second Edition, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 1996.

[5] A. A. Mitsyuk, Y. V. Kotylev, Layered layouts for software systems visualization using nested petri nets, in: V. Itsykson, A. Scedrov, V. Zakharov (Eds.), Tools and Methods of Program Analysis, Springer International Publishing, Cham, 2018, pp. 127–138.

[6] I. A. Lomazova, Nested petri nets - a formalism for specification and verification of multi-agent distributed systems, Fundam. Inform. 43 (2000) 195–214. URL: http://dx.doi.org/10.3233/FI-2000-43123410. doi:10.3233/FI-2000-43123410.

[7] T. Freytag, M. Sänger, Woped - an educational tool for workflow nets, in: BPM (Demos), volume 1295 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, p. 31.

[8] N. Nikitina, A. Mitsyuk, Carassius: A simple process model editor, Proceedings of ISP RAS 27 (2015) 219–236.

[9] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, K. Jensen, Cpn tools for editing, simulating, and analysing coloured petri nets, in: W. M. P. van der Aalst, E. Best (Eds.), Applications and Theory of Petri Nets 2003, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 450–462.

[10] L. Cabac, M. Haustermann, D. Mosteller, Renew 2.5 - towards a comprehensive integrated development environment for petri net-based applications, in: Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings, 2016, pp. 101–112. URL: http://dx.doi.org/10.1007/978-3-319-39086-4_7. doi:10.1007/978-3-319-39086-4_7.

[11] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, W. M. P. van der Aalst, Prom 6: The process mining toolkit, Proc. of BPM Demonstration Track 615 (2010) 34–39.

[12] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: BPM, volume 4714 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 328–343.

[13] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Process and deviation exploration with inductive visual miner, in: BPM (Demos), volume 1295 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, p. 46.

[14] C. W. Günther, A. Rozinat, Disco: Discover your processes, in: BPM (Demos), volume 940 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012, pp. 40–44.

[15] A. Burattin, M. Cimitile, F. M. Maggi, Lights, camera, action! business process movies for online process discovery, in: Business Process Management Workshops, volume 202 of *Lecture Notes in Business Information Processing*, Springer, 2014, pp. 408–419.

[16] M. de Leoni, S. Suriadi, A. H. M. ter Hofstede, W. M. P. van der Aalst, Turning event logs into process movies: animating what has really happened, Software and Systems Modeling 15 (2016) 707–732.