International Journal of Parallel, Emergent and Distributed Systems

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/gpaa20

Modelling the behavioural component of the emergent parallel processes of working with graph databases using Petri net-tools

Stanislav Velykodniy, Zhanna Burlachenko & Svitlana Zaitseva-Velykodna

To cite this article: Stanislav Velykodniy, Zhanna Burlachenko & Svitlana Zaitseva-Velykodna (2021): Modelling the behavioural component of the emergent parallel processes of working with graph databases using Petri net-tools, International Journal of Parallel, Emergent and Distributed Systems, DOI: 10.1080/17445760.2021.1934836

To link to this article: https://doi.org/10.1080/17445760.2021.1934836

Published online: 30 May 2021.

INTERNATIONAL JOURNAL OF PARALLEL, EMERGENT AND DISTRIBUTED SYSTEMS
https://doi.org/10.1080/17445760.2021.1934836

Modelling the behavioural component of the emergent parallel processes of working with graph databases using Petri net-tools

Stanislav Velykodniy [a], Zhanna Burlachenko [a] and Svitlana Zaitseva-Velykodna [b]

[a]Automated Environmental Monitoring Systems Department, Odessa State Environmental University, Odessa, Ukraine; [b]Informatics Department, Odessa State Environmental University, Odessa, Ukraine

ABSTRACT

The article focuses on the transformation of the behavioural UML diagrams modelled by the authors, which reflect the emergent manifestations of the life cycles of working with graph databases (GDB), to models in the form of Petri nets. Such a transformation process was carried out for the diagrams of objects, states, sequences, and activities. The extended UML 2.5 notation and Enterprise Architect 14.0 CASE tool were used in the formation of the project architecture. Modelling of the emergent parallel processes of working with GDBs in the form of a Petri net might initially seem to make the sequence more complicated for the designer to perceive than while using UML modelling, in which the relationships between entities on a diagram are obvious to the user. Conversely, the representation of behavioural models in the form of Petri nets facilitates the reflection of other important emergent points, in particular, by demonstrating objects relationship, dynamic information updates, components, or other necessary information concerning parallel GDB design.

ARTICLE HISTORY
Received 27 February 2021
Accepted 23 May 2021

KEYWORDS
Graph database; Petri net; UML diagram; behavioural modelling; structure; rendering

Introduction

Currently, there is a substantial number of software tools designed for the performance of a considerable range of specialised tasks. Some of them are limited to a single industry, while others are widely applied. Still, the general trend is directed at the specialisation of software products. A large number of software products with a wide range of modelling characteristics are developed, with BRL-CAD being one of such CAD systems.

BRL-CAD is a specialised open-source cross-platform system. It is a powerful 3D CAD for modelling three-dimensional objects using CSG methods. This CAD includes an interactive geometric editor, parallel raytracing, rendering, and geometric analysis.

Since the operation of BRL-CAD software is based on a large number of parallel processes, the best methodology for modelling synchronous interaction that can perform such modelling and analyse the emergent parallel processes and flows is the Petri nets tool.

Additionally, since early 2020, the topic of the article has unexpectedly gained relevance. Such an increase in relevance is associated with the implementation of quarantine measures and a significant slowdown in business processes since in the fall of 2019, there was still no model of the impact of the coronavirus pandemic on both business and economy.

CONTACT Stanislav Velykodniy velykodniy@gmail.com Automated Environmental Monitoring Systems Department, Odessa State Environmental University, Lvivska str., b. 15, off. 130. 65016, Odessa, Ukraine

This article has been republished with minor changes. These changes do not impact the academic content of the article.

Supplemental data for this article can be accessed here. https://doi.org/10.1080/17445760.2021.1934836

Considering the current circumstance caused by quarantine restrictions, most business segments cannot afford to order a new commercial CAD from the developer, which is where open graphical modelling systems gain relevance, as exemplified by BRL-CAD. The primary point of interest is the process of modelling parallel streams of graphic information using Petri nets to analyse free and unapplied resource channels of the BRL-CAD system.

Materials and methods

Petri nets are a modelling tool widely used in computer science, industry, engineering, project management, and business process reengineering. In this article, the authors use the methods introduced in the articles which articulated the basic aspects of the transformation of UML models to a different behavioural representation. This method is addressed in the research by [1], in which equivalent ratios of UML diagrams and corresponding OETPN models were applied. Such models have proven efficient in research materials related to the behaviour of urban transport and its deployment schemes, in particular, the connection of methods of representation of objects and diagrams of machines. Such diagrams are widely used to process behavioural modelling materials.

The methods of describing possible states illustrate behaviour over long life cycles [2]. The materials of these studies use methods of machine diagrams verification in UML notation. At the same time, the methods of Petri nets analysis are well suited for research materials related to the parallel behaviour of systems. The methods allowing to switch from models in UML representation to materials in the form of Petri nets require thorough verification of behavioural relationships.

The research materials use formal methods of testing the behaviour of block diagrams presented in [3]. Such methods allow us to gradually solve problems related to software design. However, it is important to control the behaviour of block- and/or UML diagrams' activity, which is efficiently accomplished by Petri models with a colour representation of the required materials. In this case, the transformation methods imitate the dynamic changes of states, values, and expressions, and the resulting Petri net is verified for the correctness of its behaviour by using Coloured Petri Nets (CPN) methods.

The CPN method is an effective approach to formalising and analysing UML state diagrams. In this respect, the materials of the research by [4] suggest a method of building UML diagrams based on a returned report on software design. However, in this case, the inaccurate semantics of UML diagrams during model transformation does not allow strictly adhere to the formal semantics of Petri nets.

In the case of real-time systems, the key role is played by the methods of analysis of performance parameters. Although performance analysis is carried out following the development of the system, performance can be enhanced at the early stages. In this case, the quality of the system can be improved using the transformation methods presented in the materials of [5]. In this article, the analysis of modelling and analysis of real-time and embedded systems (MARTE) profile represents the performance domain in the form of UML-diagrams sequence models. The methods of transforming such diagrams into a generalised stochastic Petri net model allow maintaining the rules for presenting the meta-model and add the Atlas Transformation Language (ATL). The tool used for verifying the suggested research is the application of thematic materials from the production domain of the Kanban system.

Analytical review of sources

A distinctive component of computer graphics tasks is the processing of graph databases (GDBs), which are essentially 'ordinary' databases built on mathematical algorithms for image reconstruction based on the generated statistical coordination data. Although not every CAD system has such features, the latter is necessitated by the current trends.

It is a widely known fact that software architecture is undergoing several changes. Such a statement also applies to updating GDBs. In case it is necessary to perform an analysis of architecture, as done by the authors of the article [6], and switch back to the software, some difficulties can emerge in the automation of the return route. Analysis of the materials of the publications of the last decade has revealed a significant number of methods and approaches based on model transformation [7] to exclude non-functional details from software architectural descriptions [8]. The methods of bidirectional transformation of UML-models with maintained accessibility properties were initiated in the research by [9]. This method has been extended by using Generalized Stochastic Petri Nets (GSPN) to analyse and summarise changes made with GDB.

The basic elements of the Petri nets theory are presented in the monograph by [10]. Behavioural models reflecting the advantages of using the Petri nets toolkit without reference to a specific programming language are set out in [11].

The classic publications [12–14] are devoted to the study of behavioural and structural properties and methods of analysing the Petri nets. Essentially, [14] contains an entire section devoted to modelling a parallel system based on graphs, which makes it possible to develop UML modelling based on this study. In [15] discusses stochastic networks and their application to performance modelling, as well as the modelling of high-level networks using logic programming techniques. Over time, the research works of these authors got embodied in the methodology [16] and further in the series of studies [17–19] which established the basis for the development of a new direction in behavioural modelling of parallel processes and facilitated the publication of this special issue of 'International Journal of Parallel, Emergent and Distributed Systems' journal.

Considering the research works whose findings directly prompted the authors to come up with the idea of transforming behavioural UML diagrams and their representation in the form of Petri nets when working with GDBs, the literature sources below are suggested to focus on.

The research by [20] suggests metrics for measuring scalability based on similarity metrics and complexity between two business process models. To this end, a Petri net was used to model the business process. Similarity metrics are measured by behavioural and structural Petri models, as well as by the complexity of the control flow, to measure the complexity of the Petri net. Considering the experiment illustrated, this article uses 4 arithmetic Petri models to measure scalability metrics.

The research by [21] suggests a new model of the approach to the elimination of deviations between the activities of the process model and the documentation in the generated event logs in the representation of the logic of Petri nets in information systems. Simultaneous transition pairs between selections are built based on process trees.

In the research by [22], a method that detects the inefficient behaviour of the Human–Computer Interaction (HCI) system was developed, assuming that at least one optimal HCI strategy is known. This method is based on tracking user behaviour logs using a Java application. The article also presents an interesting illustration of the application of this method to the environment task by using simulated realistic user behaviour.

The research by [23] suggests a set of requirements for modelling and documenting complex systems. Considering that most modelling methodologies, such as SSAD, UML, and BPMN, are used to illustrate complex systems, there is a need is their extension and refinement by using Petri nets models. This article also compares system block diagrams with Petri net representation and assesses the modelling of systems based on block diagrams and Petri nets.

Results of experimental studies

First, each model of interaction with GDB in open BRL-CAD using UML methodology with extended notation 2.5 will be presented to ensure the clarity of perception for analysts and system architects. This model representation was performed by the authors of this article in the research works by [24,25]. Subsequently, the transformation of each model from a representation in the form of a UML diagram to the form of a Petri net will be addressed.

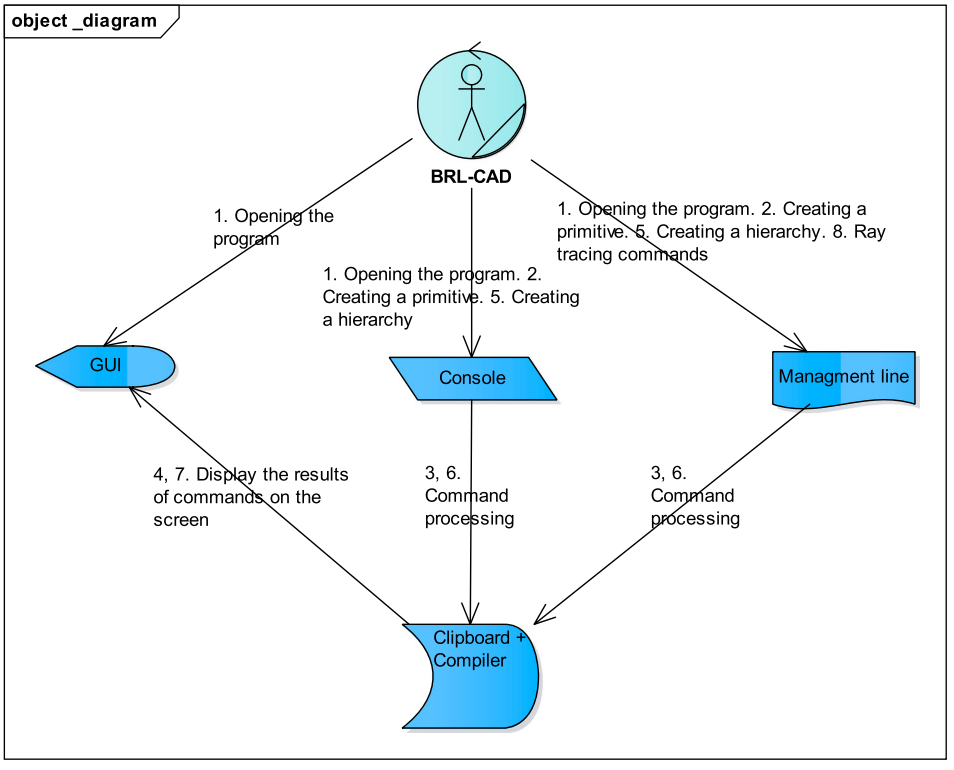

Figure 1. Object diagram for the BRL-CAD environment.

Table 1. Specification of the OEDM project environment.

Petri net element	Symbol	Function (purpose)	Position or Transition (*P* / *T*)
Activation	*AV*	Activation of input/output objects	*T*
BRL-CAD	*BC*	CAD	*P*
Clipboard + Compiler	*CC*	Processing device	*P*
Console	*CL*	Input device	*P*
Command Processing	*CP*	Command Processing	*T*
Data Recording	*DR*	GDB data recording	*T*
GUI	*GUI*	Graphic data output	*P*
Management Line	*ML*	System notification device	*P*
Output Results	*OR*	Graphic information transmission to the design	*T*

Modelling of an object diagram

The designed generalised object diagram (OD) illustrated in Figure 1 uses object entities. Before its transformation into a Petri net, a new concept that more accurately describes this diagram in a network representation – the object environment description model (OEDM) – will be introduced. This model demonstrates the interaction between abstract functional devices and software (establishment of communication links between objects) and their capabilities.

The specification of the OEDM project environment is illustrated in Table 1.

The structure of models represented in the form of Petri nets is built as follows:

$$M = \{P, T, I, O, \mu\}, \tag{1}$$

where P – net positions; T – transitions; I – input positions and transitions; O – output positions and transitions; μ – number of tokens in a specific position.

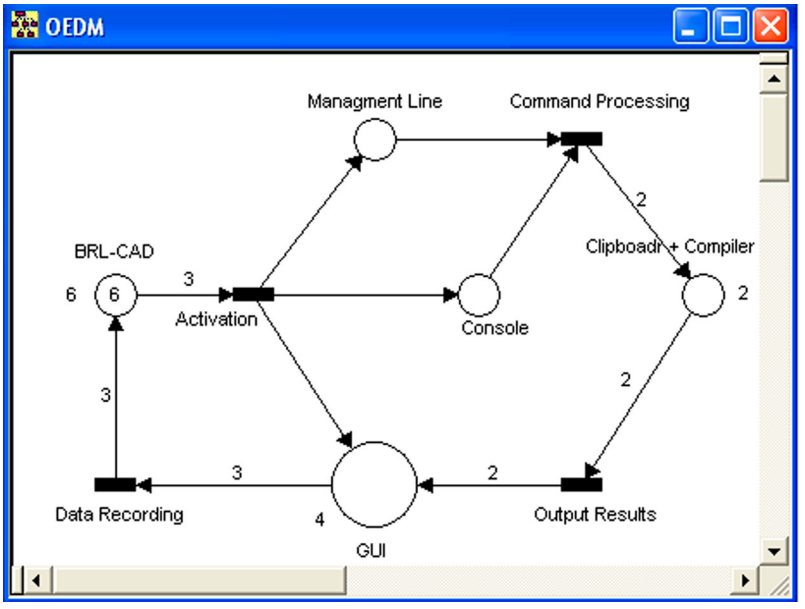

Figure 2. Network topology of the object environment description model.

Concerning OEDM, the structure (1) assumes the following form:

$$M_1 = \{P_1,\ T_1,\ I_1,\ O_1, \mu\},$$

where $P_1 = BC, CC, CL, GUI, ML$; $T_1 = AV, CP, DR, OR,$ which is described in more detail below.

$$P_1 \left\{ \begin{array}{c} BC(I,O,\mu) \left\{ \begin{array}{c} I = \{DR, DR, DR\} \\ O = \{AV, AV, AV\} \\ \mu = 6 \end{array} \right. \\ CC(I,O) \left\{ \begin{array}{c} I = \{CP, CP\} \\ I = \{CP, CP\} \end{array} \right. \\ CL(I,O) \left\{ \begin{array}{c} I = \{AV\} \\ O = \{CP\} \end{array} \right. \\ GUI(I,O) \left\{ \begin{array}{c} I = \{AV, OR, OR\} \\ O = \{DR, DR, DR\} \end{array} \right. \\ ML(I,O) \left\{ \begin{array}{c} I = \{AV\} \\ O = \{CP\} \end{array} \right. \end{array} \right\}; T_1 \left\{ \begin{array}{c} AV(I,O) \left\{ \begin{array}{c} I = \{BC, BC, BC\} \\ O = \{CL, GUI, ML\} \end{array} \right. \\ CP(I,O) \left\{ \begin{array}{c} I = \{CL, ML\} \\ O = \{CC, CC\} \end{array} \right. \\ DR(I,O) \left\{ \begin{array}{c} I = \{GUI, GUI, GUI\} \\ O = \{BC, BC, BC\} \end{array} \right. \\ OR(I,O) \left\{ \begin{array}{c} I = \{CC, CC\} \\ O = \{GUI, GUI\} \end{array} \right. \end{array} \right\}.$$

The network topology of OEDM is illustrated in Figure 2.

'*BC*' position is the initial condition for CAD operation. It contains 6 tokens and activates through the '*AV*' transition the operation of input/output devices (positions: '*CL*,' '*ML*,' '*GUI*'). With the launch of '*BC*,' the first 3 tokens move to the specified positions. Further, the condition for the command processing ('*CP*' transition as an event) is the transfer of information from the '*ML*' and '*CL*' positions in the form of entered commands. Generally, as seen from Figure 2 illustrating Petri net, OEDM describes the standard steps a user takes when working with CAD:

- opening the '*BC*' programme, which leads to the activation of '*AV*';
- creating a primitive using the '*CL*' and '*ML*' console commands;
- processing of '*CP*' commands and processing of graphic information '*CC*';
- the output of the '*OR*' command results to the '*GUI*' screen;
- creating and maintaining the hierarchy of '*DR*' design.

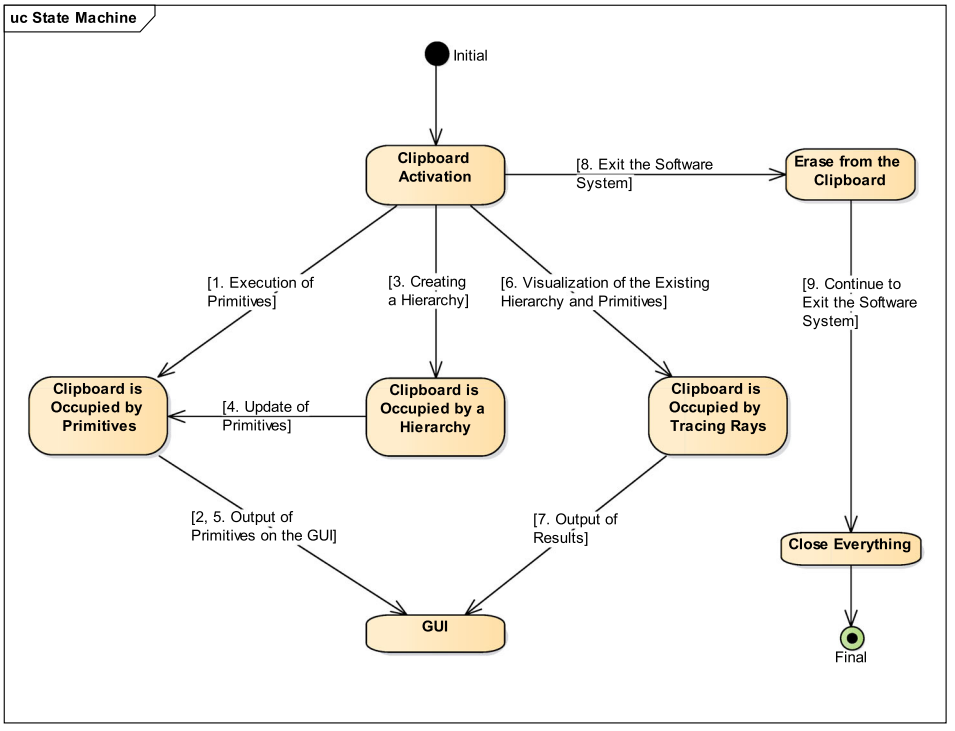

Figure 3. State machine diagram of GDB for the BRL-CAD environment.

As seen from the simulation results, the '*GUI*' position can contain 2, 3, or 4 tokens, which means a step-by-step update of the graphic information built on the screen. Therefore, the condition for the update is the execution of the ray tracing command.

OEMs will be useful in cases of a necessity to assess the consequences of changes since they illustrate which objects interact with each other and how they are labelled. When making changes to objects in the form of tokens, it immediately becomes evident which other surrounding objects will be affected. For instance, the first transition in the given net will be the opening of a programme that moves one token to three objects at once.

The second transition of tokens is based on the processing of '*CP*' graphic information to create '*CC*' primitives. The abovementioned processes transfer to the third step – output of the results of the '*OR*' command on the '*GUI*' screen. The standard step to take after building graphic objects is their visualisation. To this end, the abovementioned operation of ray tracing control should be carried out. After that, it is possible to build a hierarchy of rays and create more complex graphic objects by maintaining '*DR*' to the GDB of BRL-CAD ('*BC*').

During the implementation of OEDM for the BRL-CAD project, the structure of the Petri net was determined by describing the key conditions and events in the form of positions and transitions. Labelling in the form of tokens allowed to track the set communication connection and to model the graphic information update on the user's screen.

State machine Diagram Modelling

The designed state machine diagram (SMD) concerning the GDB of the BRL-CAD environment is illustrated in Figure 3. This diagram demonstrates the possible states for a GDB. The initial UML diagram was developed by the authors of this article in [24].

Table 2. Specification of state description model.

Petri net element	Symbol	Position or Transition (*P* / *T*)
Clipboard Activation	*CA*	*P*
Close Everything	*CE*	*P*
Continue to Exit the Software System	*CES*	*T*
Creating a Hierarchy	*CH*	*T*
Clipboard is Occupied by a Hierarchy	*COH*	*P*
Clipboard is Occupied by Primitives	*COP*	*P*
Clipboard is Occupied by Tracing Rays	*COT*	*P*
Erase from the Clipboard	*EC*	*P*
Expectations of Further Interaction	*EFI*	*T*
Execution of Primitives	*EP*	*T*
Exit the Software System	*ES*	*T*
GUI	*GUI*	*P*
Interface Interaction	*II*	*T*
Output of Primitives on the GUI	*OP*	*T*
Output of Results	*OR*	*T*
User	*U*	*P*
Update of Primitives	*UP*	*T*
Visualisation of the Existing Hierarchy and Primitives	*VE*	*T*

To perform the transformation of the state machine diagram to the state description model (SDM) in the representation of Petri net, the specification of the correspondence of the states of the UML diagram to the network elements will be developed (Table 2).

The SDM will be presented by the designations of the elements of Petri net in Table 2 and the structure (1), as follows:

$$M_2 = \{P_2,\ T_2,\ I_2,\ O_2, \mu\},$$

where $P_2 =$ CA, CE, COH, COP, COT, EC, GUI, U; $T_2 =$ CES, CH, EFI, EP, ES, II, OP, OR, UP, VE.

$$P_2 \left\{ \begin{array}{l} CA(I,O) \left\{ \begin{array}{c} I = \{II\} \\ O = \{CH, EP, ES, VE\} \end{array} \right. \\ CE(I,O) \left\{ \begin{array}{c} I = \{CES\} \\ O = \{\} \end{array} \right. \\ COH(I,O) \left\{ \begin{array}{c} I = \{CH\} \\ O = \{UP\} \end{array} \right. \\ COP(I,O) \left\{ \begin{array}{c} I = \{EP, UP\} \\ O = \{OP\} \end{array} \right. \\ COT(I,O) \left\{ \begin{array}{c} I = \{VE\} \\ O = \{OR\} \end{array} \right. \\ EC(I,O) \left\{ \begin{array}{c} I = \{ES\} \\ O = \{CES\} \end{array} \right. \\ GUI(I,O) \left\{ \begin{array}{c} I = \{OP, OR\} \\ O = \{EFI\} \end{array} \right. \\ U(I,O,\mu) \left\{ \begin{array}{c} I = \{EFI\} \\ O = \{II\} \\ \mu = 3 \end{array} \right. \end{array} \right\}; T_2 \left\{ \begin{array}{l} CES(I,O) \left\{ \begin{array}{c} I = \{EC\} \\ O = \{CE\} \end{array} \right. \\ \left. \begin{array}{c} CH \\ EP \\ ES \\ VE \end{array} \right\} I = \{CA\}, O \left\{ \begin{array}{c} CH = \{COH\} \\ EP = \{COP\} \\ ES = \{EC\} \\ VE = \{COT\} \end{array} \right. \\ EFI(I,O) \left\{ \begin{array}{c} I = \{GUI\} \\ O = \{U\} \end{array} \right. \\ II(I,O) \left\{ \begin{array}{c} I = \{U\} \\ O = \{CA\} \end{array} \right. \\ \left. \begin{array}{c} OP(I) = \{COP\} \\ OR(I) = \{COT\} \end{array} \right\} O = \{GUI\} \\ UP(I,O) \left\{ \begin{array}{c} I = \{COH\} \\ O = \{COP\} \end{array} \right. \end{array} \right\}.$$

The Petri net built according to the configured structure is illustrated in Figure 4.

The main logical chain of this model is that the user action is always performed using the '*CA*' position and completed with the '*EFI*' transition, which symbolises the start and the completion of working with the environment. It should be noted that the work with a software product will start with the interaction with its '*II*' interface, i. e. start, and finish with deactivation or closing of the product – '*CE*.'

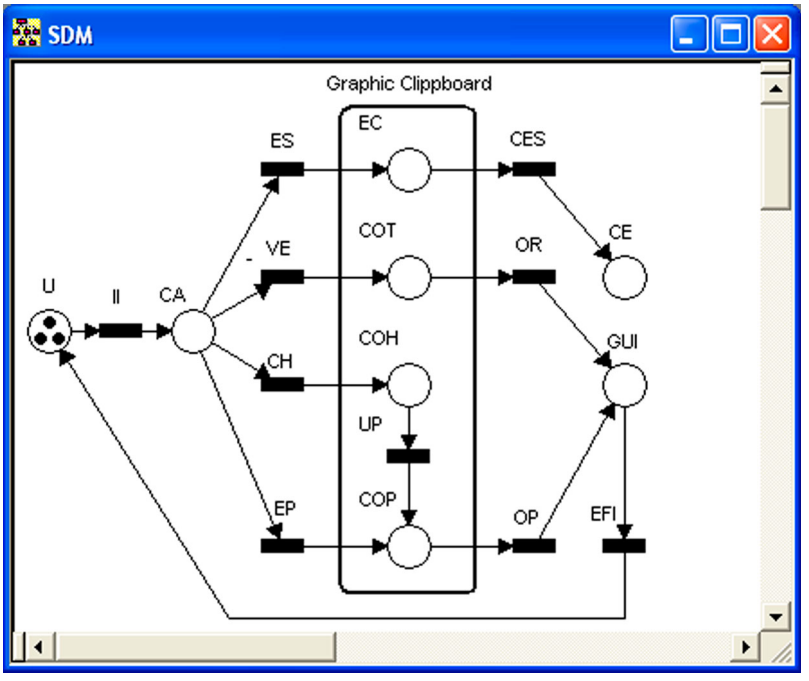

Figure 4. Network topology of the state description model.

During the start of the operation, the Graphic Clipboard and the compiler are activated. The clipboard is a stack for the shape processing commands that are currently being processed. The compiler is a special module that processes command inputs and builds the graphical result on the screen after the processing of these commands is completed.

In following a certain methodology of working with GDB, the first step is to create a primitive by using the option of output into an updated project in a previously designed GDB. To this end, the command to create a certain type of '*EP*' primitive, its size and position (if the position is not specified, the primitive is created at the origin of the coordinates) is output from the console. The further process is as follows: the clipboard receives certain information and saves it as long as it is processed by the compiler ('*COP*' position). After the compiler has processed all the information on the current command, in case the result is correct, the data is output ('*OP*' transition) to the screen ('*GUI*' position) and recorded in ROM. After several primitive figures are created, they are used to build a hierarchy ('*CH*' transition), i. e. to create more complex figures ('*COH*' position).

As mentioned above, the conditions for creating more-complex shapes include their number (more than two) and collisions (intersections). As soon as the command to build a complex shape is transferred to a clipboard, the compiler processes it. However, at this stage, not only the results are displayed on the screen, but also the replacement and update of the data from ROM ('*UP*' position) is performed, as follows: '*COP*' – '*OP*' – '*GUI*'. After the commands are completed, the clipboard is automatically cleared.

The next standard step after the creation of complex shapes is their visualisation ('*VE*' transition). To this end, it is necessary to perform rays racing ('*COT*' position). For this purpose, the pattern should be set from the console or the command line (which constitutes the object and a dozen other standard physical constants), after which one should go to a special panel of the command line and perform tracing. The further procedure is similar: 'clipboard-compiler-display' ('*COT*' – '*OR*' – '*GUI*'). The only difference is that the result is not saved in ROM, being only displayed on the screen. Another BRL-CAD module called Archer is used to record the visualisation.

The work with the GDB is completed by the visualisation, and therefore, the following actions are described by an arc ('*CA*' – '*ES*' – '*EC*' – '*CES*' – '*CE*'), which is illustrated in the Table 2, and do not require additional explanation.

It should be separately noted that the given software product has some features that allow it to be distinguished from others in the line of CAD systems. There is no such option as 'Save Project / Upload Project.' Any action is processed by the compiler and automatically saved in ROM in the form of a GDB.

Such a system organisation is associated with both advantages and shortcomings.

Advantages: in cases of the emergency closure of the environment, all information is saved, which is impossible to overestimate.

Shortcomings: in case a mistake is made, it can be corrected only by removing the entire projected figure and making changes to the hierarchy – another major shortcoming, the correction of which is an established research prospect for the authors of this article.

After the transformation of the designed UML state diagram to the SDM in the form of a Petri net and considering the analysis of its content, it is possible to draw intermediate but fairly unexpected conclusions. In particular, since BRL-CAD, which later became open, was built for the needs of the US military, it can be assumed that the original implementation of the principles of working with the environment described was the key point of interest.

Sequence diagram modelling

The sequence diagram (SD) is designed to illustrate in which sequence the programmer performs actions when working with the GDB of the BRL-CAD environment. In such diagrams, messages are arranged from top to bottom, with each being labelled with a respective name.

Since the SD designed by the authors of the article [24] essentially describes itself (Figure 5) in the illustrated methodology of the sequence, it will be omitted in this article to avoid duplication of the

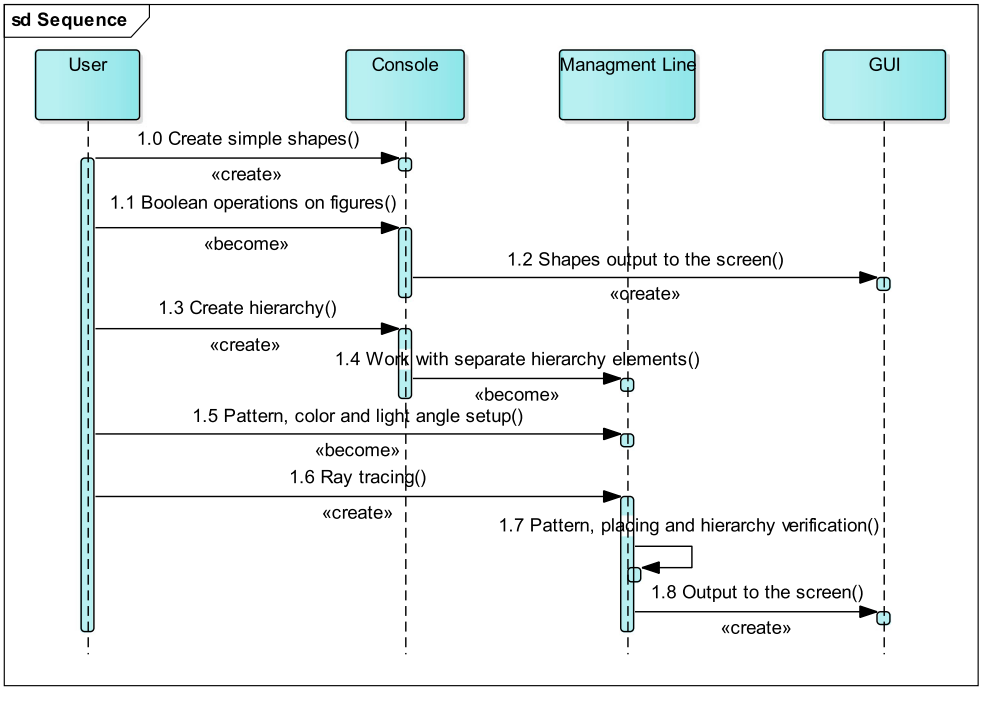

Figure 5. Sequence diagram for working with GDB in BRL-CAD environment.

Table 3. Specification of development sequence model.

Petri net element	Symbol	Position or Transition (*P* / *T*)
Console	*CL*	*P*
Graphic User Interface	*GUI*	*P*
Management Line	*ML*	*P*
User	*U*	*P*
Create simple shapes	1.0	*T*
Boolean operations on figures	1.1	*T*
Shapes output to the screen	1.2	*T*
Create hierarchy	1.3	*T*
Work with separate hierarchy elements	1.4	*T*
Pattern, colour and light angle setup	1.5	*T*
Ray tracing	1.6	*T*
Pattern, placing and hierarchy verification	1.7	*T*
Output to the screen	1.8	*T*

illustrated information. If necessary, SD can be used to demonstrate arguments and some control information. Similarly, it is possible to show the reflective messages that the object sends to itself – in such a case, the arrow of the message points to the same lifeline.

The definition of an SD itself illustrates the sequence of actions. However, with this structure being analysed by the performer (programmer), a problem of misconception arises that only such a sequence of actions should be performed as recommended by the diagram, which is not the case. Every message that starts from the lifetime of a 'User' object can be executed first at any time. This feature questions the suitability of SDs as a tool for modelling processes that can start from one object and transfer to others in the multiple structures of options.

Such special modelling of the representation of the information flows connecting other system resources can maintain the tools of the labelled Petri nets, in which the fulfilment of the condition is represented by a token in a position corresponding to this condition. The model that will reflect such features of the development of the graphical representation of information will be referred to as the development sequence model (DSM). The correspondence of the UML-diagram entities to the designation of the elements of the Petri net is illustrated in Table 3.

As can be seen, the designation of transitions (events) of the DSM corresponds to the numbering of messages on the SD.

After the transformation of the SD into the DSM is as established in Table 3, the DSM represented in the form of a Petri net shall be illustrated as a structure in accordance with (1):

$$M_3 = \{P_3,\ T_3,\ I_3,\ O_3, \mu\},$$

where $P_3 = CL; GUI, ML, U; T_3 = 1.0; 1.1; 1.2; 1.3; 1.4; 1.5; 1.6; 1.7; 1.8.$

$$P_3 \left\{ \begin{array}{c} CL(I,O) \left\{ \begin{array}{c} I = \{1.0; 1.1; 1.3\} \\ O = \{1.2; 1.4\} \end{array} \right. \\ GUI(I,O) \left\{ \begin{array}{c} I = \{1.2; 1.8\} \\ O = \{\} \end{array} \right. \\ ML(I,O) \left\{ \begin{array}{c} I = \{1.4; 1.5; 1.6; 1.7\} \\ O = \{1.7; 1.8\} \end{array} \right. \\ U(I,O,\mu) \left\{ \begin{array}{c} I = \{\} \\ O = \{1.0; 1.1; 1.3; 1.5; 1.6\} \\ \mu = 8 \end{array} \right. \end{array} \right\}; T_3 \left\{ \begin{array}{c} \left. \begin{array}{c} 1.0 \\ 1.1 \\ 1.3 \end{array} \right\} (I,O) \left\{ \begin{array}{c} I = \{U\} \\ O = \{CL\} \end{array} \right. \\ \left. \begin{array}{c} 1.2(I) = \{CL\} \\ 1.8(I) = \{ML\} \end{array} \right\} O = \{GUI\} \\ \left. \begin{array}{c} 1.4(I) = \{CL\} \\ \left. \begin{array}{c} 1.5 \\ 1.6 \end{array} \right\} I = \{U\} \\ 1.7(I) = \{ML\} \end{array} \right\} O = \{ML\} \end{array} \right\}.$$

The Petri net built according to the presented structure is illustrated in Figure 6.

The network illustrated in Figure 6 is used to simulate the fundamental steps of the 'User' object, which must be taken when working with GDB. The number of initial tokens (8) corresponds to the number of messages on the SD. All objects (tokens) displayed in the 'GUI' move to the '*DR*' transition

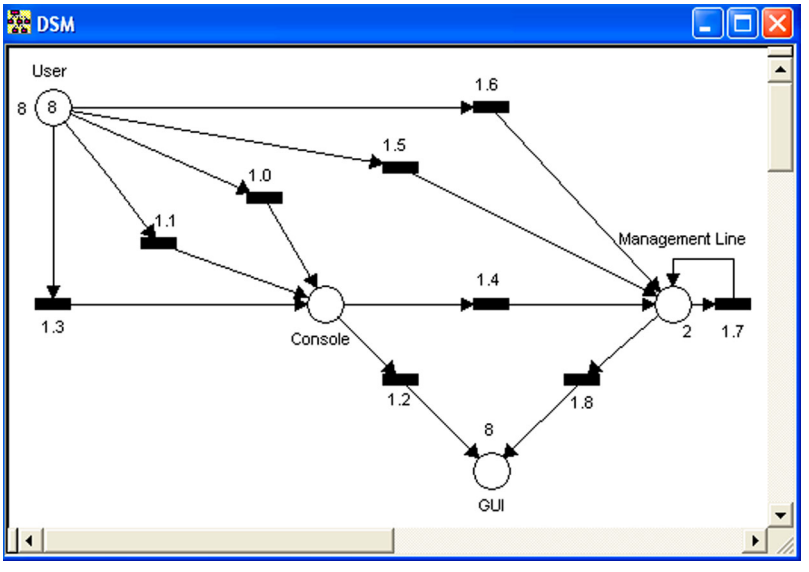

Figure 6. Network topology of the development sequence model.

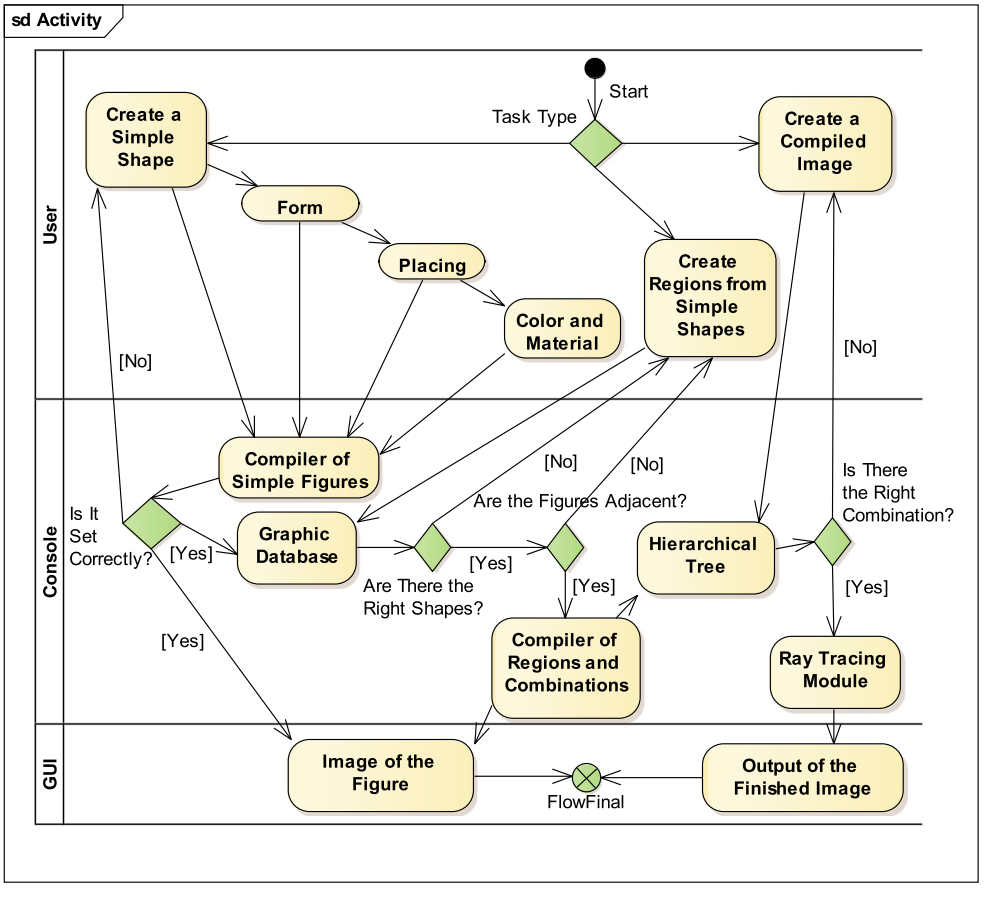

Figure 7. Activity diagram of GDB building in BRL-CAD environment.

Table 4. Specification of a dynamic activity model.

Petri net element	Symbol	Position or Transition (*P* / *T*)
Are the Figures Adjacent?	*AFA*	*P*
Additional Processing	*AP*	*T*
Are There the Right Shapes?	*ATR*	*P*
Check	*C*	*P*
Create a Compiled Image	*CCI*	*T*
Colour and Material	*CM*	*P*
Compiler of Regions and Combinations	*CRC*	*P*
Create Regions from Simple Shapes	*CRS*	*T*
Compiler of Simple Figures	*CSF1, CSF2*	*T, T*
Create a Simple Shape	*CSS*	*T*
Form	*F*	*P*
Graphic Database	*GDB*	*T*
Graphic User Interface	*GUI*	*P*
Hierarchical Tree	*HT*	*T*
Image of the Figure	*IF*	*T*
Is It Set Correctly?	*ISC*	*P*
Is There the Right Combination?	*ITR*	*P*
No 1–4	*N1, N2, N3, N4*	*T, T, T, T*
Output of the Finished Image	*OFI*	*T*
Placing	*P*	*P*
Ray Tracing Module	*RTM*	*P*
Start	*S*	*T*
Simple Figure	*SF*	*P*
Task Type	*TT*	*P*
User	*U*	*P*
User Decision	*UD*	*P*
Yes 1–3	*Y1, Y2, Y3*	*T, T, T*

and then to the '*BC*' position, as illustrated in Figure 2. The given DSM does not reflect these actions, as they have no relation to user behaviour.

Thus, in the executed subunit, a DSM in the form of a Petri net was built following the developed SD for working with GDB in the BRL-CAD environment. The model illustrates the presentation of the user's step-by-step behavioural scenario.

Modelling of the activity diagram

An activity diagram (AD) is a special diagram reflecting the decomposition of information flows of activity on the relevant tracks. Activity means the specification of the behaviour demonstrated in the form of coordinated sequential or parallel execution of subordinate elements: embedded activities and individual actions interconnected by flows directed from the out of one node to the input of another.

In our case, the AD was designed to create a GDB in the BRL-CAD environment. This diagram was elaborated by the authors of this article in [24] and is presented in Figure 7.

The behavioural UML diagram shall be transformed into a dynamic activity model (DAM) represented by a Petri net. For this purpose, the entities included in the AD will be presented in the form of Table 4 providing a list of DAM elements, designations of the respective elements and information flows, as well as the redefinition of activities to the form of representation of the Petri net: position or transition (*P* / *T*).

Next, let's proceed to the design of a mathematical model of the DAD structure in the form of a Petri net, in accordance with the following representation (1):

$$M_4 = \{P_4,\ T_4,\ I_4,\ O_4, \mu\},$$

where $P_4 = 15 = AFA, ATR, C, CM, CRC, GUI, F, ISC, ITR, P, RTM, SF, TT, U, UD$; $T_4 = 18 = AP, CSF1, CSF2, CRS, CSS, CCI, GDB, HT, IF, N1, N2, N3, N4, OFI, S, Y1, Y2, Y3$.

$$
P_4\left\{\begin{array}{l}
AFA(I,O)\left\{\begin{array}{c} I=\{Y2\} \\ O=\{N3,Y3\}\end{array}\right. \\
ATR(I,O)\left\{\begin{array}{c} I=\{GDB\} \\ O=\{N2;Y2\}\end{array}\right. \\
C(I,O)\left\{\begin{array}{c} I=\{CCI\} \\ O=\{HT\}\end{array}\right. \\
CM(I,O)\left\{\begin{array}{c} I=\{AP\} \\ O=\{CSF2\}\end{array}\right. \\
CRC(I,O)\left\{\begin{array}{c} I=\{Y3\} \\ O=\{IF\}\end{array}\right. \\
GUI(I,O)\left\{\begin{array}{c} I=\{IF;OFI\} \\ O=\{\}\end{array}\right. \\
F(I,O)\left\{\begin{array}{c} I=\{CSS\} \\ O=\{AP;CSF1\}\end{array}\right. \\
ISC(I,O)\left\{\begin{array}{c} I=\{CSF1;CSF2;CRS\} \\ O=\{N4;GDB\}\end{array}\right. \\
ITR(I,O)\left\{\begin{array}{c} I=\{HT\} \\ O=\{N1;Y1\}\end{array}\right. \\
P(I,O)\left\{\begin{array}{c} I=\{AP\} \\ O=\{CSF2\}\end{array}\right. \\
RTM(I,O)\left\{\begin{array}{c} I=\{Y1\} \\ O=\{OFI\}\end{array}\right. \\
SF(I,O)\left\{\begin{array}{c} I=\{CSS\} \\ O=\{AP;CSF1\}\end{array}\right. \\
TT(I,O)\left\{\begin{array}{c} I=\{B\} \\ O=\{CCI;CRS;CSS\}\end{array}\right. \\
U(I,O,\mu)\left\{\begin{array}{c} I=\{\} \\ O=\{S\} \\ \mu=3\end{array}\right. \\
UD(I,O,\mu)\left\{\begin{array}{c} I=\{N1;N2;N3;N4\} \\ O=\{CCI;CRS;CSS\} \\ \mu=2\end{array}\right.
\end{array}\right\};\ T_4\left\{\begin{array}{l}
AP(I,O)\left\{\begin{array}{c} I=\{F;SF\} \\ O=\{CM;P\}\end{array}\right. \\
\left.\begin{array}{c} CCI \\ CRS \\ CSS\end{array}\right\} I=\{TT;UD\}, O\left\{\begin{array}{c} CCI=\{C\} \\ CRS=\{ISC\} \\ CSS=\{F;SF\}\end{array}\right. \\
\left.\begin{array}{c} CSF1(I)=\{F;SF\} \\ CSF2(I)=\{CM;P\}\end{array}\right\} O=\{ISC\} \\
GBD(I,O)\left\{\begin{array}{c} I=\{ISC\} \\ O=\{ATR\}\end{array}\right. \\
HT(I,O)\left\{\begin{array}{c} I=\{C\} \\ O=\{ITR\}\end{array}\right. \\
IF(I,O)\left\{\begin{array}{c} I=\{CRC\} \\ O=\{GUI\}\end{array}\right. \\
\left.\begin{array}{c} N1(I)=\{ITR\} \\ N2(I)=\{ATR\} \\ N3(I)=\{AFA\} \\ N4(I)=\{ISC\}\end{array}\right\} O=\{UD\} \\
OFI(I,O)\left\{\begin{array}{c} I=\{RTM\} \\ O=\{GUI\}\end{array}\right. \\
S(I,O)\left\{\begin{array}{c} I=\{U\} \\ O=\{TT\}\end{array}\right. \\
Y1(I,O)\left\{\begin{array}{c} I=\{ITR\} \\ O=\{RTM\}\end{array}\right. \\
Y2(I,O)\left\{\begin{array}{c} I=\{ATR\} \\ O=\{AFA\}\end{array}\right. \\
Y3(I,O)\left\{\begin{array}{c} I=\{AFA\} \\ O=\{CRC\}\end{array}\right.
\end{array}\right\}.
$$

The designed structure is illustrated in the form of a Petri net in Figure 8.

The simulation is started from the '*U*' position, which contains three tokens. Let's consider first the basic start of the process, i. e. the creation of a simple '*CSS*' figure. To create a simple shape in the given network model, the user has not only to enter the command title, but also specify '*F*' and, if necessary, '*AP*,' '*P*' and '*CM*,' as shown in Figure 8.

After these parameters are specified, all information is transmitted to the compiler of simple figures (at the level of the physical code module, there is one compiler, which, however, will be divided for clarity into sublevels: '*CSF*1' and '*CST*2'). The specific feature of this network is the '*ISC*' position, which indicates that the condition is met and translates in such case to 'GDB,' or otherwise returns to 'UD,' where the decision is to be made by the user (two-token labelling).

Therefore, after the information is received by the compiler, such a check of the condition is performed. Assuming that the conditions are not met, such information as 'Shape is not created' will be issued at the discretion of the user, and the current command will be interrupted. If the check is passed successfully, i. e. the '*AFA*,' '*ATR*,' and '*CRC*' conditions are positively met, then '*IF*' is executed, and graphical information is displayed at the '*GUI*' position.

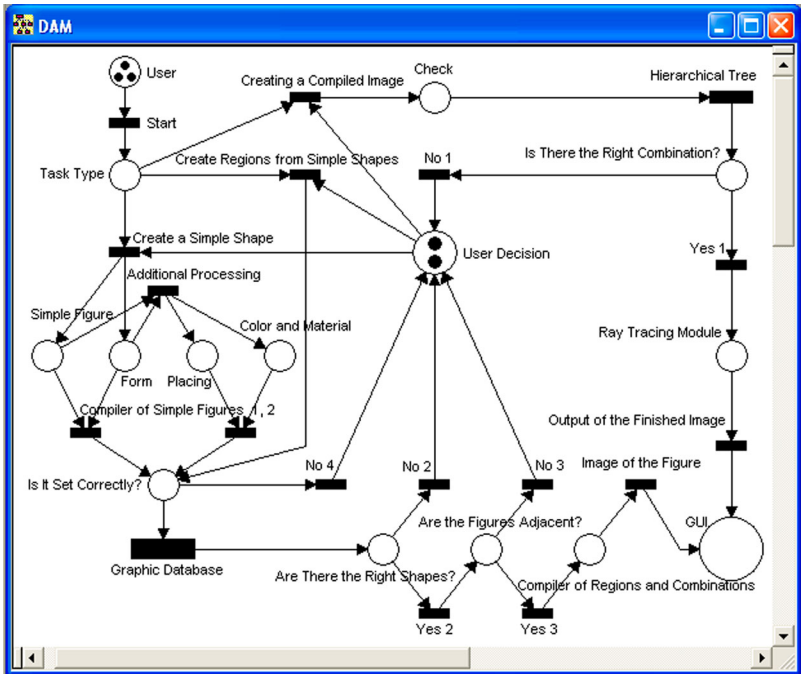

Figure 8. Network topology of the dynamic activity model.

Another '*TT*' output after the creation of simple shapes is '*CRC*.' During attempts to create regions, the process of checking the '*ISC*' condition, which is necessary for the creation of complex shapes, is launched. If all conditions are met, recording to the memory and displaying on '*GUI*' are processed.

As can be seen from the Petri net, at each unspecified stage of adoption, the decision is transmitted to the '*UD*,' which can be only reached through negative transitions 'N1 – N4' that are triggered in case of non-compliance with the relevant conditions and positions.

The '*TT*' output is a branch: '*CCI*' – '*C*' – '*HT*' – '*ITR*' – '*Y*1' which means that after creating complex hierarchical figures, it is possible to trace the 'RTM' image. Ray tracing is performed using a special module after checking the fulfilment of certain conditions. It should be added that primitives can also be traced, although such practice is not widely adopted due to its futility because, apart from complex components of figures, it is necessary to perform a check for physical constants. To ensure correct tracing, it is necessary to indicate the angle of incidence of light, the level of the transmitted light, the material from which the figure is made, etc. Provided that all these parameters are specified correctly, an '*OFI*' image that is perfectly suitable for use as a GDB component output to the '*GUI*' can be obtained.

Thus, the behavioural AD UML diagram was transformed into a DAM one with the representation of its Petri net, which allows determining states, routes, and cycles of designing GDB in BRL-CAD.

Discussion

During the observations and discussions of the process of teaching users the basics of working with GDB, this process was initially presented in the form of behavioural UML diagrams. The emergent information flows were modelled and documented using these diagrams. UML diagrams were also used to study the internal structure of the GDB. However, when diagrams are large, complex, and branched, their scenario behaviour is difficult for users to understand and hard to use to support decision-making in the design and configuration of GDBs.

It was suggested to additionally represent the behaviour during the configuration of the GDB in the form of Petri nets, which proved an appropriate suggestion. The effectiveness of the suggested models was confirmed experimentally by accelerating the process of behavioural learning the parallel running of GBD. Petri nets were proved a promising alternative to UML diagrams due to the wide range of parallel analysis and modelling capabilities. Network analytics can be used to study the emergent behavioural properties of graphical processes. This modelling strategy was also used to describe how it can support parallel GDB design processes.

Thus, modelling of work with GDB in the form of a Petri net can initially complicate for the designer the perception of the sequence as compared to modelling by using UML methodology, in which the location of links between entities is arranged on the top of the page to the bottom. Conversely, such a spatial arrangement makes it easier to reflect other important aspects that might unexpectedly arise. For example, it is possible to show the relationship of objects, dynamic updating of information on the screen, the terms of connection of a particular component, or other information related to the design of the GDB.

General results obtained

Research has shown that high rendering speed offers broad prospects for the application of BRL-CAD in a range of industries, including military, industrial or training purposes, such as design and analysis systems in mechanical engineering, mechanical components, architectural structures, chemical molecule structure, etc.

The advantage obtained over the equivalents ensures the wide scope of the application opportunities for the BRL-CAD system, ranging from modelling the structure of the atomic nucleus to galactic dimensions.

Notably, the method of transforming behavioural UML-diagrams into the Petri nets was set out in the article about achieving the set objectives,

The result of transformations of behavioural UML models illustrates that all the presented Petri nets models are scalable to each other in different propagation rates of positions and transitions increase, which gives the obtained results a substantial advantage over the equivalents in the form of linear transformations.

In illustrating the established differences with the equivalents, it can be stated that the scale growth rate determines the extent to which the expanded geo-database can be transformed into more complex models.

It was established that the solution to the outlined problems is impossible without a profound insight into the physical essence of the studied phenomena, development, and advancement of the relevant theoretical provisions, as well as the practical implementation of the achieved results and analysis of their illustrativeness.

Geometric transformation methods have long been used successfully in many industries. The newly developed methods of geometric modelling and their implementation in the systems of computer processing of graphic information play an important role in this regard since they make it possible to deal with the specifically established tasks in graphic processing.

Considering that in any CAD, the events and interaction between the system and the designer occur asynchronously and independently (there is no link between the system response and the subsequent actions of the designer, and the final decision is made by the user based on data issued by the system), for modelling and further research in this area, the use of Petri nets tools can be the best option for modelling and further research of this field.

Research has shown that project managers or users themselves develop their strategies for dealing with repetitive tasks and processes. However, organisations have difficulty in determining the extent of the effectiveness of employees' interaction with the software. Organisations typically train employees to interact with the required software. One of the best examples of such interaction is the suggested method of transforming diagrams of geo-database management. It should also be added in this regard

that the obtained Petri nets illustrate the processes of configuration and interaction with open geo-databases.

The obtained method of transformation provided recommendations for the gradual adaptation of the behaviour of operators necessary to establish the optimal way of interaction with the geo-databases with regard to the level of the mathematical competence of users and its further advancement.

General conclusions

The research works discussed should continue being carried out considering the special interest in computer graphics and due to the intensive development and implementation of CAD in various fields of production and training. Although CAD BRL-CAD is acceptable for use for an experienced designer, for a beginner or student, the process of its application can seem complicated. The materials that can be found on the World Wide Web are superficial and contain only a few dozen console commands.

A detailed analysis of the environment revealed the presence of two modules contained in the structure of CAD which can assist the potential system user to quickly design the necessary GDB. Another fundamental feature of the package is the ability to support the design and analysis of visual models based on complex objects consisting of a large set of graphical primitives.

Generally, the article achieved the set goal to model the emergent aspects of design, connection, and further improvement of GDB as a composite component of open BRL-CAD using the methodology of representing these parallel processes in the form of Petri nets.

The performed researches of models of working with GDB are of importance since they allow to improve BRL-CAD by transferring the offered Petri nets to high-level language coding and the provision of updating. After the research, it can be concluded that the strength of the system is the extraordinary speed of visualisation, ray tracer, and rendering. Comparison with analogues justifies the statement that the visualisation process is one of the fastest among the existing ones.

The prerequisite for this research was the modelling of the design framework in its representation in the form of Petri nets, which will serve as the basis for the future technological rationale for building GDB for connection to the updated open BRL-CAD.

The simulation resulted in building Petri nets based on the transformation of behavioural UML diagrams: states, sequences, activities, and, optionally, objects. The extended UML 2.5 notation and CASE tools Enterprise Architect 14.0 were used in the formation of the design architecture.

The short-term objective of the research is the real-time implementation of the obtained models. This process will take a long time, namely from 6 months to a year, which can be attributed to the fact that the ultimate goal of our transformations is to accelerate and record the time needed to train users to use BRL-CAD graphic databases.

Importantly, this idea can be further adapted by other researchers to complete the next stage of the study within the framework of the established plan to conduct the experiment and record the learning results of the two groups of users in real time. One group will be trained using the traditional approach, and the other will apply the Petri net models. Based on the numerical data, it is possible to establish the timing for triggering the Petri nets transitions.

We also plan to expand the research by introducing the tools for modelling parallel processes in the form of Sleptsov nets. The short-term objective of the research is the conversion of the developed Petri nets to Sleptsov nets.

Another research perspective is the transformation of structural UML diagrams developed by the authors of this article, including classes, components, composite structure, review of interaction, and deployment to the representation in the form of Petri / Sleptsov nets. This presentation will substantially facilitate the use and improvement of structural and static GDB models of open CAD.

Acknowledgments

The authors would like to express their gratitude to BRL-CAD corporation for the opportunity to openly use and test the source files and plugins and the provided support for the cross-platform methodology. The authors also thank James Lyle Peterson, the author of the monograph [10], which contains the basics of Petri nets theory and has been the core reading for the author of this article for more than 25 years. Additionally, the authors thank Professor Dmitry Zaitsev for providing, by personal example, the motivation to perform the research presented in the article.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Stanislav Velykodniy http://orcid.org/0000-0001-8590-7610
Zhanna Burlachenko http://orcid.org/0000-0001-8451-5527
Svitlana Zaitseva-Velykodna http://orcid.org/0000-0001-7453-8821

References

[1] Santa MM, Cuibus OP, Al-Janabi D, et al. Relations of UML and OETPN Models. Proceedings of the 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR); Cluj-Napoca, Romania; 2020. p. 1–6. DOI: 10.1109/AQTR49680.2020.9129999
[2] Lyazidi A, Mouline S. Formal verification of UML state machine diagrams using Petri nets. Proceedings of the international conference on networked systems; In: M Atig, A Schwarzmann editors. Networked systems. NETYS 2019. lecture notes in computer science, vol 11704. Cham: Springer. p. 67–74. DOI: 10.1007/978-3-030-31277-0_5
[3] Gulati U, Vatanawood W. Transforming flowchart into coloured Petri nets. Proceedings of the 3rd International Conference on Software and e-Business; 2019 Dec. p. 75–80. DOI: 10.1145/3374549.3374568
[4] Meziani L, Bouabana-Tebibel T, Bouzar-Benlabiod L. From Petri nets to UML Model: a new transformation approach. Proceedings of the 2018 IEEE International Conference on Information Reuse and Integration (IRI); 2018; Salt Lake City, UT. p. 503–510. DOI: 10.1109/IRI.2018.00080
[5] Shailesh T, Nayak A, Prasad D. An UML based performance evaluation of real-time systems using timed Petri net. Computers. 2020;9(4):94. DOI: 10.3390/computers9040094
[6] Velykodniy SS, Tymofieieva OS. Zaitseva-Velykodna SS the calculation method for indicators project estimation in the implementation of software systems re-engineering. Radio Electron Comput Sci Control. 2018;4:135–142. DOI: 10.15588/1607-3274-2018-4-13
[7] Zaitsev DA. Toward the minimal universal Petri net. IEEE Trans Syst Man Cybernetics Syst. 2014;44(1):47–58. DOI: 10.1109/TSMC.2012.2237549
[8] Zaitsev DA. Simulating cellular automata by infinite Petri nets. J Cellular Automata. 2018;13(1-2):121–144.
[9] Cortellessa V, Eramo R, Tucci M. Availability-driven architectural change propagation through bidirectional model transformations between UML and Petri net models. Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA); 2018; Seattle, WA: IEEE; 2018. p. 125–129. DOI: 10.1109/ICSA.2018.00022
[10] Peterson JL. Petri net theory and the modeling of systems. New York (NY): Prentice-Hall, inc; 1981.
[11] Bandman OL. Behavioral properties of Petri nets (a survey of the French literature). Tekhn Kibern. 1987;5:134–150.
[12] Ichikawa A, Hiraishi K. Analysis and control of discrete event systems represented by Petri nets. In: Varaiya P., Kurzhanski A.B. (eds) Discrete event systems: models and applications Vol. 103. Springer; Berlin, Heidelberg. 1988. p. 115–134. DOI: 10.1007/BFb0042308.
[13] Holloway LE, Krogh BH. Synthesis of feedback control logic for a class of controlled Petri nets. IEEE Trans Automat Contr. 1990;35(5):514–523.
[14] Murata T. Petri nets: properties, analysis and applications. Proc IEEE. 1989;77(4):541–580.
[15] Zaitsev DA, Sleptsov AI. State equations and equivalent transformations for timed Petri nets. Cybern Syst Anal. 1997;33(5):659–672. DOI: 10.1007/BF02667189
[16] Zaitsev DA. Sleptsov nets run fast. IEEE Trans Syst Man Cybernetics Syst. 2016;46(5):682–693. DOI: 10.1109/TSMC.2015.2444414
[17] Zaitsev DA. Universal Sleptsov Net. Int J Comput Math. 2017;94(12):2396–2408. DOI: 10.1080/00207160.2017.1283410
[18] Zaitsev DA. Sleptsov net computing (chapter 672). In: Khosrow-Pour M, editor. Encyclopedia of information science and technology. 4th ed. IGI-Global; USA. 2017. p. 7731–7743. DOI: 10.4018/978-1-5225-2255-3.ch672.
[19] Zaitsev DA, Li ZW. On simulating turing machines with inhibitor Petri nets. IEEJ Trans Electr Electron Eng. 2018;13(1):147–156. DOI: 10.1002/tee.22508

[20] Fauzan AC, Sarno R, Yaqin MA. Petri net arithmetic models for scalable business processes. Proceedings of the 3rd International Conference on Science in Information Technology (ICSITech); 2017; Bandung: IEEE; 2017. p. 104–109. DOI: 10.1109/ICSITech.2017.8257093
[21] Teng Y, Du Y. Qi L, et al. A logic Petri net-based method for repairing process models with concurrent blocks. IEEE Access. 2019;7:8266–8282. DOI: 10.1109/access.2018.2890070
[22] Theis J, Darabi H. Behavioral Petri net mining and automated analysis for human-computer interaction recommendations in multi-application environments. Proc ACM Human-Comput Interaction. 2019;3:1–16. DOI: 10.1145/3331155.
[23] Kim R, Gangolly J, Elsas P. A framework for analytics and simulation of accounting information systems: a Petri net modeling primer. Int J Account Inf Syst. 2017;27:30–54. DOI: 10.1016/j.accinf.2017.09.002
[24] Velykodniy SS, Burlachenko ZV, Zaitseva-Velykodna SS. Graphic databases reengineering in BRL-CAD open-source computer-aided design environment. Modeling of the behavior part. Trans Kremenchuk Mykhailo Ostrohradskyi National Univ. 2019;2(115):117–126. DOI: 10.30929/1995-0519.2019.2.117-126
[25] Velykodniy SS. The idealized models of software systems reengineering. Radio Electron Comput Sci Control. 2019;1:150–156. DOI: 10.15588/1607-3274-2019-1-14