The Development of Complex Data Structures Using Object Enhanced Time Petri Nets

Dahlia Al- Janabi
Dept.of Automation
Technical University of Cluj-Napoca
Cluj- Napoca, Romania
dahliajanabi@gmail.com

Tiberiu S. Letia
Dept. of Automation
Technical University of Cluj- Napoca
Cluj- Napoca, Romania
tiberiu.letia@aut.utcluj.ro

***Abstract*—The development of the complex data structures systems requires models that are capable of describing the systems' requirements, design, verification, implementation and testing. This paper shows the development of such systems using Object Enhanced Time Petri Nets (OETPNs) framework. OETPN can model concurrency, reaction to internal and external events, synchronization and temporal behavior with fix and dynamic delays. OETPN keeps the bipartite graph structure of the classical Petri Nets, but the unique kind of tokens in PNs are changed to be any kind of objects to represent the data, or tasks (sub OETPN) that can either be active (i.e. threads of execution) or passive. OETPNs are capable to communicate with each other through input and output channels and to distribute concurrent tasks that are moved through a computer network with a dynamical structure (i.e. Task migration) without the need to stop the execution of the other OETPN. Guards and mappings are used to control the flow of execution in order to model the creation, insertion, modification and extraction of information of complex data structures. The mappings are also used to implement the information transformation and the placement of the resulted information at the output places of the transitions. The types of the places are specified to allow setting of objects or tasks. A new OETPN dynamic structure that is best suited for the server side is introduced. A Java platform that implement such applications is described and a complete system to manage the operations of the data structures is discussed with an example.**

Keywords*— complex data structures; Object Enhanced time Petri Nets Framework; object-oriented programming and distributed programming.

I. Introduction

Most of the programming languages that are used for conceiving data structures systems and database systems are based on object-oriented programming, so it is expected from the modeling method to sustain the dynamical creation, modification, and removing of objects, the dynamic distributed concurrent execution that involves reaction to internal and external events, as well as managing complex structured information. Such method is expected to sustain the entire development process. For very complex systems, the method should sustain the models' integration into components that further include other components. These were the goals of the proposed method that is intended to be used through all the software development phases.

The UML (Unified Modeling Language [1]) and its real time extensions like MARTE [2] require a set of diagrams to cover all the different features of an application. The use for many models for the same application increases the development effort and reduce the possibility of creating an interactive development environment.

A data structure [3] models abstract objects. It implements special operation on this object, which are classified as follows:

- Creation
- Deletion
- Update or modify
- Query

While Databases [4] and database systems play an important role in everyday life, for example, banking, airlines, hotel reservations system, etc. A database is a collection of related data like facts that can be recorded.

A database has the following properties:

- Represents some aspects of the real world, and the changes of the real world affect the database.
- Describe logically coherent collection of data.
- Are designed, built, and populated with data for a specific purpose.

Object Enhanced Time Petri Nets (OETPNs) [5] are suited to analyze, to model and to synthesize complex data structures and database system. Building an OETPN starts with identifying the input and output channels. The main OETPN is called a parent, while the active object (i.e. sub OETPN) is called a child, the structure and behavior has to be specified in the requirements. In the same manner, the child can have children of its own and so on, creating a multi-level OETPN.

The definition of an OETPN is [5]:

$$OETPN = (P, T, pre, post, Inp, Out, D, \delta, Type, type, Grd, Map, A, M) \quad (1)$$

Where:

- $P = \{p1, p2, ..., p_m\}, (m \geq 0)$
- $T = \{t1, t2, ..., t_n\}, (n \geq 0)$

- *pre:* $P \times T \rightarrow \{0,1\}$ (with 0 and 1 the natural numbers, 0 denoting the lack of any link and 1 the existence of a link between p and t) is the backward incidence function.
- *post:* $P \times T \rightarrow \{0,1\}$ is the forward incidence function.
- δ is a mapping that assigns delays from a natural number set D to the OETPN transitions.
- *Inp* is the input place set.
- *Out* is the output place set.
- $D = \{\ldots, d_i, \ldots, d_j, \ldots\} \subset \mathbb{N}$ is a set representing the transition delays assigned by a mapping δ: $\{t \in T| |^{o}t| = 1\} \rightarrow D$,
- $Types = \{Class_1, Class_2, \ldots\}$ represents the set of software classes of the net,
- *Type*: $P \rightarrow Types$ is a mapping that assigns to each place a class (i.e. type),
- **Grd** and **Map** are the net guard and mapping sets,
- $\Lambda = \{\lambda_t | t \in T, \lambda_t \subseteq grd_t \times map_t\}$ is a set or relations $\lambda_t = \{(grd_t^i, map_t^i)\}$ that assigns to each transition t pairs composed of a guard and a mapping from the sets grd_t and map_t.
- **M** is the net vector marking with $M(p)$ the marking of the place p that identifies the object token oØf the type (p).

A guard of a transition t_k denoted by grd_k^i is a mapping:

$$grd_k^i: \prod_{p \in {}^{o}t} \{Domain(M(p)) \cup \phi\} \rightarrow \{true, false\} \qquad (2)$$

Where *true* and *false* are the values of the Boolean set. ϕ denotes the empty set. The set of all guards assigned to a transition t_k is denoted by grd_k, and **Grd** denotes the set of all the guards of the net.

A transition t_k mapping denoted map_k^i is:

$$map_k^i: \prod_{p \in {}^{o}t} \{Domain(M(p)) \cup \phi\} \rightarrow \prod_{p \in t^{o}} \{Domain(M(p)) \cup \phi\} \qquad (3)$$

Domain(M(p)) defined for the token that can be set in the place p the set (possible very complex and infinite) of the values assignable to object's attributes. All the mapping of a transition t_k are included in the set map_k and **Map** represents the set of all the mapping of the net.

OETPN models have the capability to model object-oriented programming features: synchronization, concurrency, asynchronous and synchronous communication, and distributed implementation. Features like association, aggregation and inheritance are also implemented at the place types. OETPN have the capability to clearly describe information transfer and task migration. The nodes of the OETPNs are the places that are used to store information, while the transitions describe the transformation of the information.

This research aims to introduce a dynamic OETPN structure that resembles list of a variable length. When a client requests an interaction with the database, a new place and a new transition are added to the list invoking the client's request as an active token (sub OETPN) thus instantiating a new server thread for each request as shown in the class diagram in fig. 1 where two clients are communicating with the server.

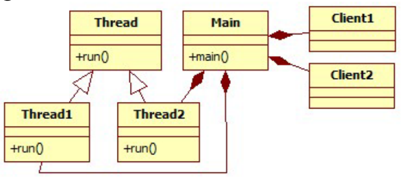

Fig. 1. A class diagram of the client-server communication

Section III provides an explanation for the OETPN complex data structures system. The structure of the proposed OETPN model is described in section V and an implementation example is presented in section VI.

II. Related Works

Petri nets (PNs) have been known for their capability to describe concurrent dynamic systems and to verify their properties. A review of the PN history is presented in [6]. Developments that concern the unification of the PNs features lead to the creation of Unified PNs [7], [6], [8]. Object PN models were proposed in [8], [9]. The tokens are implemented as objects and are considered active and passive agents that are capable to interact via shared places and shared transitions [10].

Time plays a primary role in the Enhanced Time PN [11] where the Time PN contains input and output places. A complex data structure system reacts to its input channels (modeled by input places) and sends information to its output channels (modeled by output places). An OETPN keeps the bipartite graph structure of the PN but replaces the former tokens with objects that can activate the transitions when some conditions are met. The function mappings of OETPN determines its marking. Some objects that are created in an OETPN are active and represent seperate threads of execution. When the thread execution is completed, it becomes passive. If a place contains an active or passive token and another token has to be set at the same place, the old token is turned into passive and is replaced by the new one.

A PN simulator is used for synthesizing data structures and functions that create and manipulate instances of the data structures [12].

Efficient data-structures and algorithms can be used to improve the performance of a simulator of Coloured PNs [13]. Fuzzy PN model is used for representing knowledge and behavior of an intelligent object-oriented database environment [14]. The authors of [15] utilized the concepts of PNs to model databases by interpreting various database structures. They proposed an algorithm to access the data paths between two specified nodes in a database model.

Coloured PNs are used to generate models for the execution of database transactions where algorithms to generate PNs models that specify the transactions executions are proposed in [16].

A PN based object-oriented modeling language that is considered as an object-oriented extension for the high-level PN that provides formal concurrent semantics for object-oriented models was proposed in [17]. It was used for analyzing a protocol performance.

The OETPN approach is based on PNs:

- that describe the interaction between objects,
- that model the objects' behaviors,
- whose tokens are PNs leading to nets within nets,
- have complex tokens modelling objects that have tokens leading to nets within nets.

III. OETPN Relations with Complex Structures

A. UML Component Diagram

Fig. 2 shows the UML component diagram for the distributed system that manages complex data structures. First the client requests a thread from the server and once it is granted, the clients communicates directly with the newly created thread that communicates with the database.

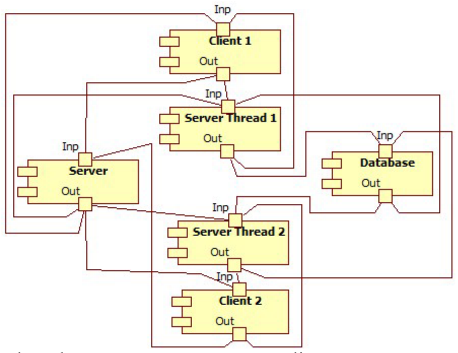

Fig. 2. Distrubuted system UML component diagram

B. State Machine Model

Fig. 3 shows the UML state machine model for the system where three threads of execution are created at the client side for creation (s11 and s12), setting (s21 and s22) and getting (s31) the information. The user uses the keyboard as an input channel and the system sends the information to the server via the TCP/ IP protocol. Theses threads are executed concurrently.

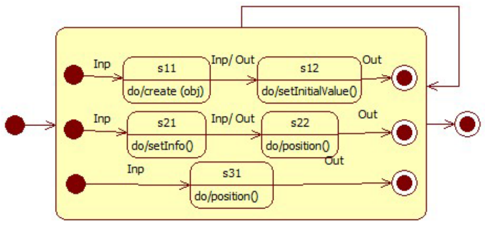

Fig. 3. UML State Machine Model for the client

C. Sequence Diagrams

Fig. 3 shows the interaction between the client and the server UML sequence diagram. First the client demands a new thread to be created at the server side, when the srever sends an answer containing the newly created thread port number and IP address, the client becomes able to communicate directly with this thread and to send the commands for creation, setting, and getting processes along with the information that is requied for each operation.

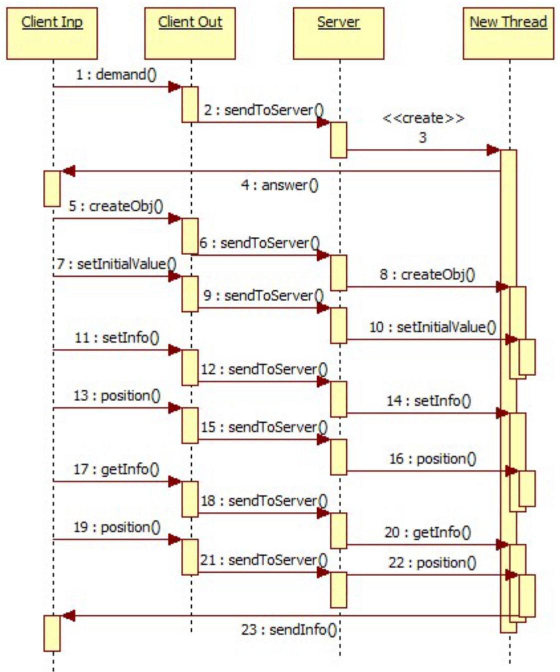

Fig. 4. The UML sequence diagram for the server

D. OETPN Model

The objects are instantiated using their class and can be used to store information calling their setter methods or retrieving information using their getter methods. Any OETPN place has a specified type and can refer to an object of this type. Generally, a class has one or more constructors that accept or not parameters. The parameters are used to create objects with the specified initial state. The calling of the setter methods with provided parameters are used to modify the object state according to the stored information and the new transmitted information. The calling of the getter methods with or without provided parameters are used to extract the required information related the transmitted information. In the case of the getter methods, the parameters can be used to specify the information that are extracted from the object and to modify the stored information with the received information.

Fig. 5. Shows the client side OETPN that allows the user to manage a complex data structures system. The places of the type String are: p10, p11, p14, p15, p17, p18, p21, p22, while the rest have the control signal type. The upper part of this OETPN model is divided into two parts, the first part is used to request a new thread of execution from the server through the input channel p2 and sends it via the output channel p3, then the OETPN waits for an answer from the server via the input channel p5, when the answer is received, the second part that it is used to create objects is enabled.

The user specifies the object type via the keyboard in p7, the mapping of t3 transmits the requested object type via the output channel p8 to the server, then the OETPN waits for the user to specify the initial value via p10 and the mapping of t4 transmites it to the server via the output channel p11.

The middle part of the model is responsible for setting information in the object that is prevoiusly created, the user can specify the positon where the information is to be stored via p14, the mapping of t5 transmits the information to the server via the output channel p15, then the user input the new information by the keyboard via p17, the mapping of t6 transmittes this information via the output channel p18 to the server.

The third part is responsible for retreiving information (get) from the server, the user inputs the position via p21 and the mapping of t7 transmits it to the server via the output channel p22.

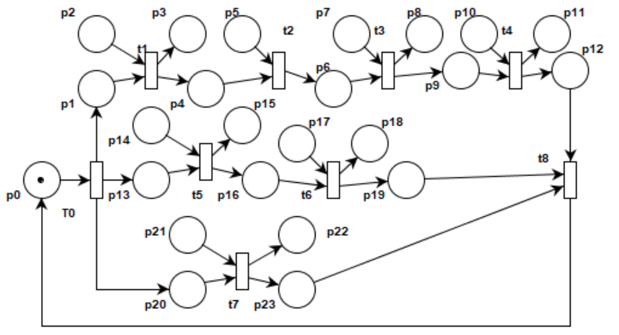

Fig.5. OETPN Client side model of a data structures management system

The server side is an OETPN of a dynamic structure that resembles a list of a variable length. The place p0 has the control signal type, the places p2, p4, p6, ..etc. have a control signal and a String type while the places p1, p3, p5, ..etc. are of an OETPN type. When the server receives a request from the client through the input channel p0, it instantly creates a new transition t1 and two new places, p1 where the new thread is created as an active token (sub OETPN) and p2 which is an output channel that sends a response to the server containing the port numbere of the newly created OETPN, then client can communicate directly with this active token, and if a second client sends a request, a new transition t2 and new places p3 and p4 are added to the server dynamic structue and a second thread is invoked from p3 as an active token for the second client and so on. Fig 6. shows the server dynamic structure.

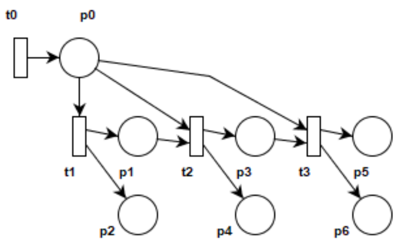

Fig.6. OETPN Server side model for the data structures management system

Fig. 7 shows the server side's active token, that can create (instantiate) an object referred by the place p0 using the transition t1, can set information in a previously created object using t2, or can extract information from the mentioned object using the transition t3. Let obj0 be this object and the type of p0 its class. The type of p1 can be of any type if the place p1 is used only to control the creation of the object obj0. The place p2 can be used for the information required for the initial value of obj0. If more than one constructor can be used for obj0 creation, its specification can be given by p1 or p2.

In a similar manner, the setting of information in obj0 performed by t2 uses the information referred by p3 and p4. One of them can be used to specify the position where the information has to be stored, and the other gives the new information that has to be stored. Obviously, the types of the places p3 and p4 should correspond to the mentioned purposes.

The transition t3 used for retrieving of the information get from the object referred by p5 the specification (i.e. the position) of the required information.

The input channels tc1, tc2, tc3, tc4 and tc5 are linked directrly to the client side OETPN output channels via p6. tc1 receives information to create a new object form the client's output channel p8, tc2 receives the initial value from the client's output channel p11, tc3 receives the position (index) from the chlient's output channel p15, tc4 receives the new information to be set form the client's output channel p18. tc5 receives the position from the client's output channel p22.

All the mentioned operations are performed by the transition mappings and they use the methods implemented in the adjacent classes assigned to places. The transition guards use the class methods to control their executions also.

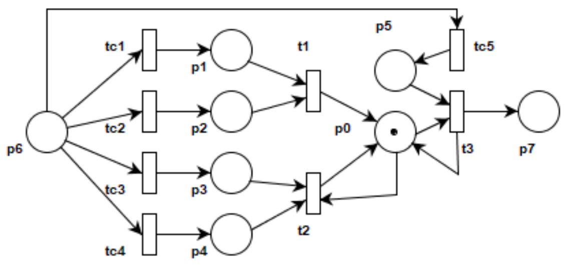

Fig.7 Server side's active token model for the data structures management system

IV. OETPN FOR TACKLING THE DATABASES (DB)

To manage a database, we need a connection. The connection can be passed throght the OETPN nodes, with this connection, a string can be used as a query to insert, update, delete and select.

V. Implementation

The implementation was performed using Java programming language. Figure 8 shows the class diagram for the OETPN model implementation (i.e. simulation). An OETPN is an instance of the OETPN class and an instance of the OETPNdata. Each OETPN has an instance of Pre class which is a two dimensional array of the input places for all the transitions, an instance of Post class which is a two dimensional array of the output places for all the transitions, the marking which is a one dimensional array of type OETPNdata that can either be sub OETPNs or an instance of the DataHndler class according to the places types, the DataHandler is a data structure (list, set, map, or DB), it has three general methods (add, remove, and delete) that can manage any type of data structures and are used by the transition's guards and map.

An instance of the Delay class which is a one dimensional array of the transitions delays, an instance of the Transition class for each transition which contains: the guard conditions, a delay that is obtained from the Delay's array according to the transition index, the transition input and output places have the marking index according to the Pre and Post, temporary marking which is a one dimensional array of type OETPNdata that reserves the tokens for the transition when one of its conditions is true to execute it.

Each OETPN has a list of guards (one guard for each transition), each guard has a list of subguards, each subguard has a list of predicates and a list of map. If all the predicates inside one subguard are true then the transition is enabled and this modules the (And) logical operation. But if we have two subguards that have the same map stored in two different map lists. So if one of these predicates is true, then the map is executed eitherway, and this modules the (OR) logical operation. . If more than one sub guard is true, the first one found is taken into consideration, and if there are conflict transitions, the one with the lowest index wins the lottery.

The Mapping class contains the functions which are used to validate predicates and loop over the subguards and set the mapping in the output places of the transitions as well as creating a new sub OETPN. Also, the Mapping class is responsible for calculating the tokens after executing any transition and update the marking.

The parent and the child OETPNs are executed concurrently on different threads. The parent and the child can communicate through the marking of the token that is set at the parent place where the child is created. The parent creates the structure, the guards and the mapping of the child, when the child execution ends, its marking is accessible to the parent along with the DataHndler objects. Each parent has a one dimensional array of instances of SubPetriInfo, each instance holds the information concerning each child so the parent is able to initiate or terminates the sub OETPN's thread according to the parent's guards, map and the parent place index from which it is started.

Every OETPN must have a unique name, a network port and an IP address so OETPNs can communicate with each other. The DataOverNetwork represents the messages that are exchanged by OETPNs. The NetworkListener class is responsible for starting a thread that listens to the network for the input data from other OETPNs.

The executer algorithm is in the OETPN class run method, it runs with the period of 1time unit or when data is received in the input channels, when a 1 time unit elapses, the delay is decreased by one for every transition in the execution list that only holds the enabled transitions, and when the delay reaches 0, the transition is enabled.

Also, a DataOverNetwork instance has to be initiated to send the data over network to another OETPN if the output place of the transition is an output channel. OETPNs can send and receive passive tokens (sub OETPNs) and objects through their input and output channels respectively. The passive token contains the complete data of the sub OETPN where it is an instance of the OETPNdata class, it has its own structure (pre and post) and behavior (guards conditions). An instance of the DataOverNetwork is created while defining the guard of the sender OETPN, it sends a one dimensional array of OETPNdata containing the passive sub OETPN or an object at the place index that is the input channel of the recipient OETPN along with its network port and IP address. If the output place of a transition is an output channel, the Mapping class opens a new socket with an IP address and the port form the DataOverNetwork instance, and then serializes the instance and sends it.

The recipient OETPN compares the received marking form the network with its own, if the place index contains null, then that place remains the same, if not, it replaces the old token with the one that is received from the network. If the received token is a passive sub OETPN, it is executed when it passes one of the recipient OETPN transitions that has the guard condition to do so. This ensures that the recipient OETPN continues working and there is no need to shut it down, so only the sub OETPN is turned to passive and is replaced by the new one.

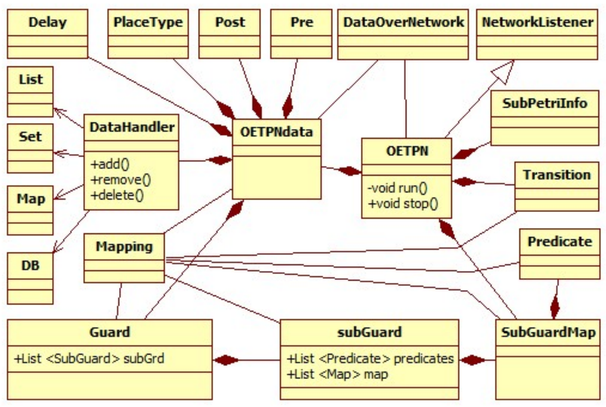

Fig. 8. Class diagram of the OETPN model

The OETPNdata has another constructor for creating the dynamic OETPN structure that is linked to the NetworkListener class, when the dynamic OETPN receives a request form a client, it creates a new transition and a new place that are linked together as a list as shown in fig. 6 thus by updating the Pre and Post instances accordingly by adding the newly added transition and places. As soon as the new place of type OETPN is created, a new thread is started with a sub OETPN that is linked to the client's input and output channels. The sub OETPN directly manages the data structure that is linked to this dynamic OETPN and performs the creation, set, and get processes as requested by the client.

VI. Implementation Example

This system can be used for a railway system. The railway agent client side OETPN is shown in fig. First, the agent sends a request to create a new thread to the dynamic OETPN server that is shown in fig. 6, when the request is granted and a new thread (sub OETPN) is created as shown in fig. 7, the agent is now able to create a new train, and assembles it using the set to add wagons and get to remove wagons. The train is a list object as shown in fig. 9, the train's header p0 stores the train information (i.e. id, model number, city, etc.). The places starting from p1 until the end of the list are the train wagons, whenever a new wagon is added, the list adds a new transition to connect the new place with the previous one and a new transition to connect the new place with p0. The transitions tc1, tc2, and tc3 are used for a direct access to the places. When the agent wishes to delete a place (wagon), the transition that has the place as an input is deleted as well along with the transition that connects it with p0, the transition that has the deleted place as an output will be connected to the next place in line as demonstrated in fig.9 with the red line.

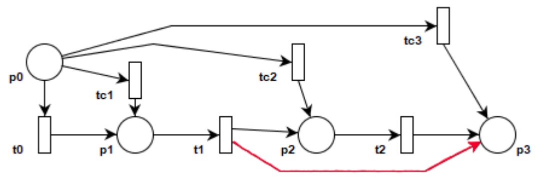

Fig. 9 A train list object

VII. Conclusions

Complex data structures systems essential in our everyday life where everything concern us is managed by it (citizen records, banking accounts, university records, etc.). OETPN can be used to model complex data structure systems as the data structure objects can be manipulated by it. An example to manage an information system for railway is provided in this paper. We have extended the OETPN to have an active tokens (sub OETPNs) and data structure objects that can be anything like (list, sets, maps, databases, etc.). The DataHandler class is responsible for providing the create, set and get methods for each data structure type.

The dynamic OETPN structure at the server side that resembles a list of a variable length is able to interact with many clients and creates a thread as an active token for each client to directly manage the data structure according to the client requests and sends back the active token's port number and IP address when it is first created so the clients can store them and modify the mapping for the transitions that are responsible for sending tokens over the network via the output channels. For future work, we intend to connect the OETPN with UML and create OETPN models accordingly.

Acknowledgment

The authors would like to thank the Technical University of Cluj-Napoca, the Automation and Computer Science department for which none of this would have been possible without their support.

References

[1] OMG Unified Modeling Language. http://www.omg.org/spec/UML/2.5/ ,2015.

[2] OMG. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, http://www.omgmarte.org/, 2011.

[3] Peter Brass, "Advanced Data Structures", Cambridge university press, ISBN-13 978-0-521-88037-4, 2008.

[4] Elmasri Navathe, "Fundamentals of Database Systems sixth Edition", Addision Wesley, ISBN-13: 978-0-136-08620-8, 2011.

[5] Tiberiu Letia, Dahlia Al- Janabi, "Object Enhanced Time Petri Nets", IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), IEEE Xplore, DOI: 10.1109/AQTR.2018.8402743, 2018.

[6] J. R. Silva, J. A. S. P. Miralles, A. O. Salmon, P. M. Gonzalez del Foyo, "Introducing Object-Oriented in Unified Petri Net Approach", ABCM Symposium Series in Mechatronices vol.4, pp. 451-459, 2010.

[7] C. Ghezzi, D. Mandrinoli, S. Morasca and M. Pezze. " A Unified High Level Petri Net Formalism for Time Critical Systems", IEEE Transactions on Software Engineering, DOI: 1.1109/32.67597, 1991.

[8] T. S. Letia and A. O. Kilyen. "Unified Enhanced Time Petri Net models for development of the reactive applications", 3rd International Conference on Event-Based Control, Communicatio and Signal Processing (EBCCSP), DOI: 10.1109/EBCCSP.2017.8022831, 2017.

[9] T. Miyamota, S. Kumagi, "A survey of objects-oriented Petri Nets and Analysis methods", IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, DOI:10.11093/ietfec/e88-a.11.2964, 2005.

[10] V. A. Bashkin, "Modular Nets of Active Resources", Automatic Control and Computer Sciences, DOI: 10.3103/S0146411612010026, 2012.

[11] Tiberiu Letia, Dahlia Al- Janabi, "Object Enhanced Time Petri Nets", 3rd International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP), DOI: 10.1109/EBCCSP.2017.8022831, 2018.

[12] Reggie Davidrajuh, "Efficient Data Structures for a New Petri Net Based Simulator", European Modelling Symposium, DOI: 10.1109/EMS.2014.10, 2014.

[13] Kjeld H. Mortensen, "Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator", Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, pp. 57-75, 2001.

[14] Burcian Bostan-Korpeoglu, Adnan Yazici, "A fuzzy Petri net model for intelligent databases", Elsevier Science Publishers, DOI: 10.1016/j.datak.2006.08.003, 2006.

[15] Gurdeep Singh Hura, Harpreet Singh, N. K. Nanda, "Some design aspects of databases through Petri net modeling", IEEE Transactions on Software Engineering, vol. SE-12, issue 4, pp. 505-510, 1986.

[16] Kees M. van Hee, Natalia Sidorova, Marc Voorhoeve, Jan Martijn van der Werf, "Fundamenta Informaticae – Concurrency Specification and Programming (CS&P), vol. 93, issue 1-3, pp. 171-184, 2009.

[17] F. Vale de Azevedo Guerra, J. C. Abrantes de Figueiredo and D. D. S. Guerrero, "Protocol Performance Analysis Using a Timed Extension for an Object Oriented Petri Net Language", Elsevier, Electronic Notes in Theoretical Computer Science, vol.130, pp. 187-209, 2005.