FEDERAL STATE AUTONOMOUS EDUCATIONAL INSITUTION FOR HIGHER EDUCATION
NATIONAL RESEARCH UNIVERSITY
«HIGHER SCHOOL OF ECONOMICS»

Faculty of Computer Science

Anton Rigin

Name, Surname

Симулятор расширения сетей Петри с ссылочной семантикой и семантикой данных на основе Renew

Russian title

Reference and Data Semantic-Based Simulator of Petri Nets Extension with the Use of Renew Tool

English title

Coursework (Term Project)
Field of study 09.04.04 «Software engineering»
Program: System and Software Engineering

Student

_____

Anton Rigin

Supervisor

_____

Sergey Shershakov,
Senior Lecturer at School of Software Engineering,
Research Fellow at PAIS Lab, Faculty of Computer Science, HSE

Moscow, 2020

Abstract

The complexity of the software and other systems is constantly growing, which is even more enhanced by concurrency of processes in the system, so modeling and validating of such systems is necessary in order to eliminate the failures. Because of the concurrency, traditional finite state machines (FSMs) are not sufficient for solving this problem regardless whether the FSM is deterministic or non-deterministic. One of the most well-known formalisms for solving this problem is the Petri net and its modifications such as the colored Petri net, the reference net and others. However, the Petri nets and their mentioned modifications do not support modeling of the persistent data usage in the system. The solution is db-net which is the formalism consisting of the three layers: the control layer which is represented by the modified colored Petri net, the data logic layer which is represented by queries for retrieving the persistent data and actions for updating (modifying) this data and the persistence layer which is represented by the relational database, its schema and constraints for keeping the persistent data in the consistent condition. However, software tools which use the db-net formalism are not available in public. In the described work the software simulator for db-nets is developed. The simulator is developed as a plugin for the Renew (the Reference Net Workshop) software tool which is built as a collection of plugins written in Java. This also allows to support the existing Renew's reference semantics. The SQLite embeddable relational DBMS is used as a base tool for implementing the db-net's persistence layer. In the paper the theoretical foundations and the architecture of the developed simulator are described. The results of the work can be used in the research projects where modeling the complex software system, which uses the persistent data, is necessary, regardless whether the project has an academic or industry-oriented purpose. This includes the research projects of the Laboratory of Process-Aware Information Systems at the HSE University's Faculty of Computer Science (PAIS Lab). Moreover, the results are expected to be used in the paper's author's master thesis based on the described term project.

Key Words: db-net; Petri net; Renew; relational DBMS; persistent data; software modeling; software validation.

Table of Contents

Introduction

In the current days, the complexity of the software and other systems is constantly growing. One of the most important constituents of such complexity is concurrency of processes performed by system since it brings a lot of uncertainty and non-deterministic behavior.

This creates a problem that traditional finite state machines (FSMs) are not enough efficient for modeling and validation of these concurrent systems regardless whether the FSM is deterministic or non-deterministic. The amount of the possible FSM's states is growing significantly, and system modeling is becoming impossible for being interpretable by human or for conducting research using the existing computing power.

One of the most popular modeling formalisms which solves this problem is Petri net which was invented by Carl Adam Petri in 1939 [1]. It allows to represent concurrent system as directed bipartite graph which consists of such vertices as places and transitions and such nodes as arcs. Tokens are used to represent the system's resources and their distribution across the net's places called marking is used to represent the current state of the system [1] [2].

Although the Petri nets are useful technique to model the concurrent software systems and the problem of significant growth of the FSM graph's nodes with the system complexity increase is solved in them, there are still problem that the Petri net may become too large for being understandable or even infinite if the data types with the very large or infinite ranges of possible values are used in the software system. The solution is the colored Petri net which supports data types called "colors", arc expressions, guard expressions and other useful tools. They firstly were described by Kurt Jensen in his article in 1997 [3] and then in the textbook by Kurt Jensen and Lars M. Kristensen in 2009 [4].

The colored Petri net can be used for modeling the concurrent software systems with the data types which can contain any number of possible values. Also there exists yet another formalism based on the Petri net which is called reference Petri nets or simply reference nets. The reference nets allow to use references to objects as the tokens, including even references to other nets. One of the well-known software tools implementing the reference nets is the Renew (the Reference Net Workshop) [5].

Even though the Petri nets and their modifications can model the complex concurrent software systems' behavior, they cannot easily model working with data in the persistent database. The solution called db-nets was proposed in 2017 by Marco Montali and Andrey Rivkin [6]. A db-net is a formal model which consists of three layers: the control layer (the modified colored Petri net), the data logic layer (queries and actions which allow the control layer to retrieve and manipulate the data in the database) and the persistence layer (the relational database schema

together with constraints which declare the data consistency rules). The last two layers and the modifications of the colored Petri net used in the control layer allow to model working with the data in the persistent database while the control layer allows to model the control flow of the system as well as working with the local (non-persistent) data. Therefore, the db-net solve the problem of working with the persistent data in the model [6]. The schema of the db-net layers is shown on the Fig. 1.

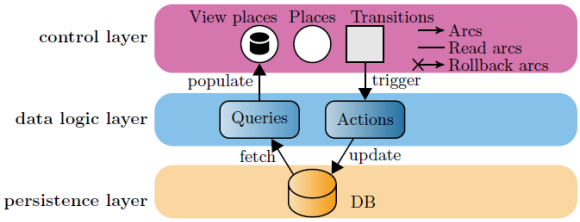

Fig. 1. The db-net structure [7].

Moreover, the db-nets can be built with usage of the reference semantics since the reference data type can be potentially considered as the ordinary data type (color) in the colored Petri net. It allows to use complex data types with large values as tokens in the net.

Although the db-nets can improve the quality of modeling the concurrent complex software systems and their validation, especially those which use the persistent data, their software implementations are not available in open sources, so no analogues of the developed in the current work software product are available in public. The aforementioned Renew tool is built as the collection of plugins written in Java and it is open-source, so extending this tool in order to support simulating the db-net run seems to be an appropriate solution and it forms the main part of the project.

The developed software simulator provides the ability to validate possible behavior of the designed complex concurrent software system even if we need to consider the persistent data used by the system. This can be used for modeling and validating the behavior of the real safety-critical software systems as well as for the further researches. The potential users of such a simulator involve the research staff of the Laboratory of Process-Aware Information Systems at the HSE University's Faculty of Computer Science (PAIS Lab) [8]. This simulator is also expected to be used in research applied to the real industrial software systems (for example, for the financial software systems) which is expected to be conducted within the paper's author's master thesis that should be completed and defended in 2021.

The purpose of the project is to develop the program (the software simulator), which supports the db-net formalism, based on the Renew open-source software tool, together with the existing Renew's reference semantics. This simulator should meet the functional requirements which are as follows.

1) The program shall allow user to model the db-net control layer using the Renew tool's graphical user interface (GUI) elements for net's interactive drawing.
2) The program shall allow user to create the database schema at the db-net persistence layer using the Renew tool's GUI.
3) The program shall allow user to declare queries and actions for the db-net data logic layer using the Renew tool's GUI.
4) The program shall allow user to simulate the modeled db-net's run using the Renew tool's GUI.
5) The program shall allow user to save the modeled db-net using the Renew tool's GUI.
6) The program shall allow user to open the previously saved modeled db-net using the Renew tool's GUI.

The developed software product consists of program (source code and executables) which meets the requirements listed above and documentation in the form of the Javadoc documentation and the current report.

Chapter 1. Theoretical Foundations

Section 1.1. Petri Nets

The Petri net is a powerful tool for modelling the processes and behavior of concurrent software systems invented by Carl Adam Petri in 1939 [1]. The Petri nets are used by different researchers for a long time since they provide wide range of the concurrent software modelling abilities. It may be useful to build other models based on them. Firstly, it should be helpful to mention the Petri net definition. Petri nets can be represented as the directed bipartite graph (bigraph) where the vertices (nodes) are places and transitions and the edges are arcs connecting the places and transitions.

Formally, the Petri net *N* is the triple $N = (P, T, F)$ [2], where:

1) P is the finite set of places of the net N;
2) T is the finite set of transitions of the net N;
3) $P \cap T = \emptyset$;
4) $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net N.

Petri nets support modelling behavior due to tokens which form markings. Conceptually, marking is a distribution of tokens across places. Formally, a marking M is a function $M: P \to \mathbb{N}$. This function maps each place into the number of tokens which are contained by this place (if there are no tokens in the place at the given moment, then this function's value is equal to 0 at the corresponding moment) [2]. The token represents the resource of the system. The whole marking represents the state of the system modelled by the Petri net.

Petri nets allow to model the concurrent systems avoiding need to depict all possible states which may constitute extremely large set or even infinite set which is difficult or impossible for people to comprehend.

The example of the Petri net for modelling the mutual exclusion is shown on the Fig. 2.

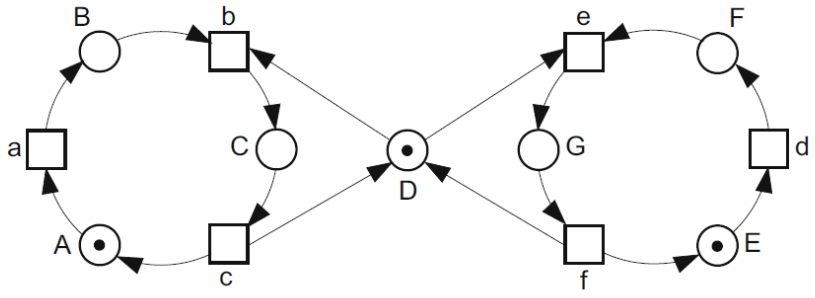

Fig. 2. The example of Petri net for modelling the mutual exclusion [2].

There are several modifications of the Petri nets for solving the specific problems.

Section 1.2. Colored Petri Nets

The colored Petri net is the modification of the Petri net which allows to define and use the typed (“colored”) tokens. Each color represents the data type. In this case arcs may have type constraints in their arc expressions in order to prevent the transmission of token of unsupported (for the given arc) token. These nets firstly were described by Kurt Jensen in his article in 1997 [3] and then in the textbook by Kurt Jensen and Lars M. Kristensen in 2009 [4].

The example of colored Petri net for modelling the simple transport protocol is shown on the Fig. 3.

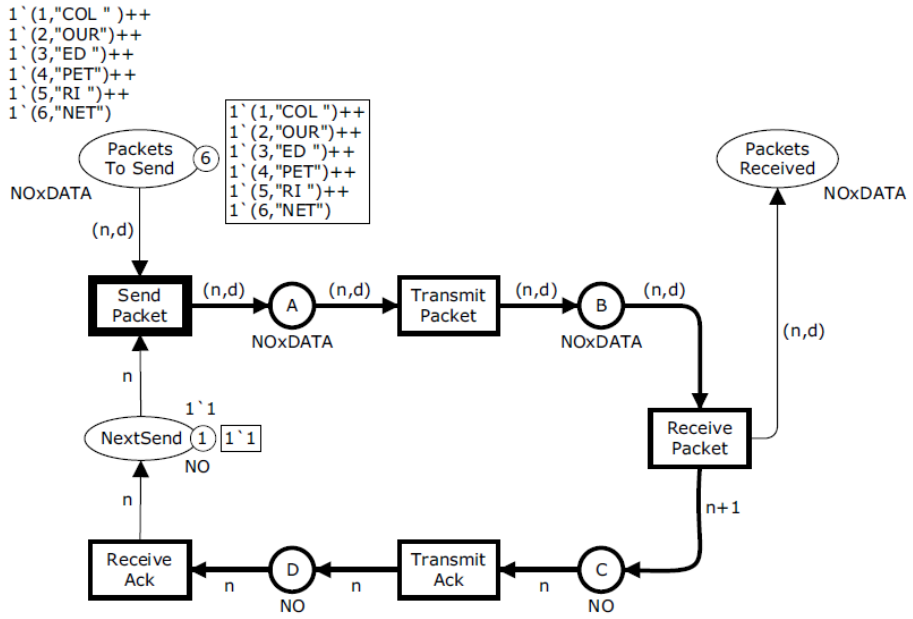

Fig. 3. The example of colored Petri net for modelling the simple transport protocol [4].

Section 1.3. Reference Petri Nets

The reference Petri net is the modification of the Petri net which have reference semantics rather than value semantics [9]. The tokens in such nets are references to objects (for example, to other Petri nets – this concept called hierarchical Petri nets, nets within nets or nested nets) [5] [9].

The example of hierarchical reference Petri net is shown on the Fig. 4.

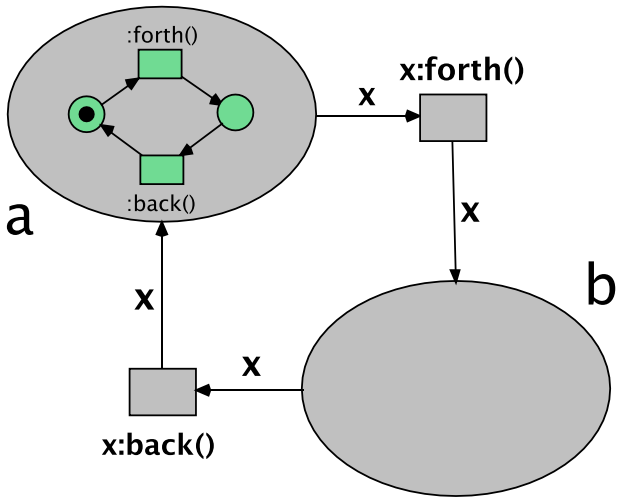

Fig. 4. The example of hierarchical reference Petri net [10].

Despite the mentioned Petri net modifications are useful in many cases, they do not allow to model working with the persistent data in the database. The possible solution for this problem is the db-net formalism which is discussed as the main topic of this work further.

Section 1.4. DB-nets

The db-net is a formal model which consists of three layers: the control layer (the modified colored Petri net), the data logic layer (queries and actions which allow the control layer to retrieve and manipulate the data in the database) and the persistence layer (the relational database schema together with constraints which declare the data consistency rules). The last two layers and the modifications of the colored Petri net used in the control layer allow to model working with the data in the persistent database while the control layer allows to model the control flow of the system as well as working with the local (non-persistent) data. Therefore, the db-net solve the problem of working with the persistent data in the model. The db-nets were proposed in 2017 by Marco Montali and Andrey Rivkin [6].

Section 1.4.1. Persistence Layer

The persistence layer of the db-net formal model is a relational database schema together with constraints which declare the data consistency rules. The database schema consists of relational table declarations (names of relational tables and types of their fields). The consistency rules are presented as boolean expressions (which may be evaluated to be equal to “true” or “false”). If after performed update action any of the consistency rules are evaluated as equal to

"false", then the database is inconsistent, and the rollback of the last update action is performed [6].

In the db-net's software implementation the persistence layer can be represented as a database in one of the relational database management systems (RDBMSs).

Section 1.4.2. Data Logic Layer

The data logic layer of the db-net formal model is a set of queries which may be used on the persistence layer for retrieving the persistent data together with a set of actions which may be performed on the persistence layer for updating (modifying) the persistent data [6].

The query in the data logic layer is a relational query for retrieving the data from the RDBMS. The action in the data logic layer is a sets of added facts (table rows) and deleted facts. During performing an action, rows from the set of added facts are inserted into the corresponding database tables and rows from the set of deleted facts are removed from them. Each added or deleted fact consists of the table name and the columns' variables for the inserted or removed row. The variable may be a simple variable (in this case it is replaced by a real value in the db-net's control layer, an external input variable (in this case it is replaced by the data inputted by a user) and a randomly generated fresh variable (in this case it is randomly generated during the action performing) [6].

Section 1.4.3. Control Layer

The control layer of the db-net formal model is a colored Petri net with the following modifications [6]:

1) In addition to traditional Petri net places there view places may exist which allow to retrieve the persistent data from the db-net's persistence layer using the query from the data logic layer assigned to the corresponding view place [6].
2) Actions from the data logic layer may be assigned to transitions. If such a transition fires (executes), then its assigned action is performed for modifying the data in the persistence layer's database [6].
3) In addition to traditional Petri net arcs there read arcs and rollback arcs may exist [6].
    a. The read arcs allow to connect view places with transitions in order to transmit the data retrieved from the persistence layer in the form of tokens [6].
    b. The rollback arcs allow to define the direction of the tokens' movement in the case when the rollback of the transition's action is applied after its performing because of emerged data inconsistency in the db-net's persistence layer [6].

The example of the db-net's control layer is shown in the Fig. 5. The left two circles depict the view places while other circles represent the traditional Petri net places. The text near the view

places is the corresponding query assignment. The lines connected to the view places show read arcs. The rectangles depict the transitions, the text pieces on these rectangles are the corresponding actions assignments. The arrows are the traditional Petri net arcs except of the arrow from the "Awake" transition to the "Stalled tickets" place which is the rollback arc.

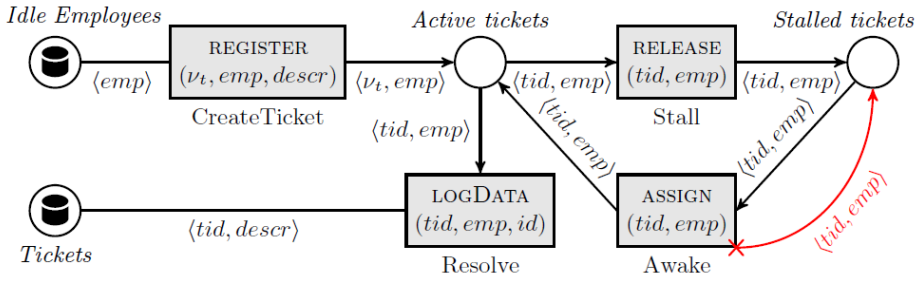

Fig. 5. The example of the db-net's control layer [6].

The part of another db-net's control layer example together with the persistence layer's relational database schema for the taxi booking information system is presented on the Fig. 6.

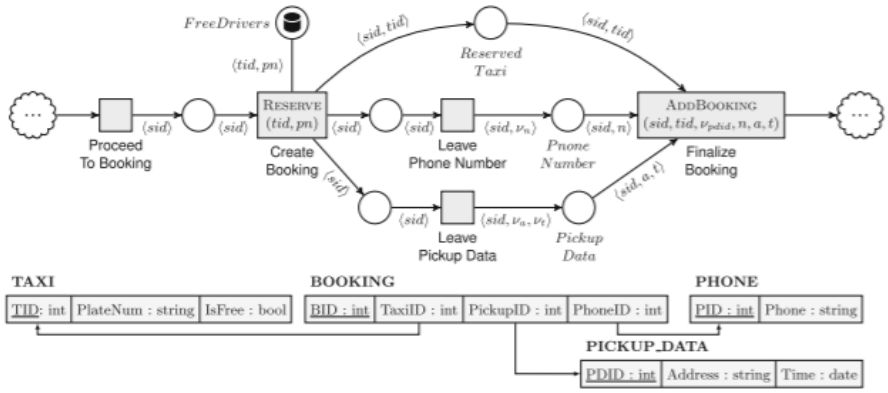

Fig. 6. The part of the db-net's control layer example together with the persistence layer's relational database schema for the taxi booking information system [6].

The db-nets are used as the basic formal model for the software tool developed in the current work.

Chapter 2. Technical Solution

Section 2.1. The Renew Software Tool

The Renew (the Reference Net Workshop) is an open-source software tool for modelling the reference and colored Petri nets. The Renew is a plugin-based tool which means that each piece of its code and each class of this tool belongs to one of its plugins. All the Renew plugins are written in the Java programming language and use the Apache Ant as a build system as well as the whole Renew software tool. The Renew software tool is being maintained at the Theoretical Foundations Group of the Department for Informatics of the University of Hamburg, Germany [5].

The main Renew plugin is the *Loader* plugin which loads all other plugins in the corresponding Renew build during the Renew tool running start. Other base Renew plugins include but not limited to the *Simulator* plugin (for the Petri net model simulation), the *Formalism* plugin (for parsing and compiling the Petri net and its constituents as formal models), number of the GUI plugins and others [5].

Adding new plugin is possible through implementing the subclass of the *de.renew.plugin.PluginAdapter* class from the *Loader* plugin and adding several build and configuration files as well as adding the information about the new plugin into the main Apache Ant *build.xml* build file [5].

Section 2.2. The Developed DB-nets Renew Plugin's Overall Structure

In this project the db-nets Renew plugin is developed in order to add into the Renew tool the ability to work with the db-nets together with the existing Renew's reference semantics.

The db-nets plugin is developed using the existing Renew base plugins as examples. The db-nets plugin's Java classes extend and use the classes of the following base Renew plugins:

1) *CH (DrawPlugin)* – plugin with drawing utils;
2) *Formalism* – plugin with classes which parse and compile the Petri net and its constituents as formal models;
3) *GUI* – plugin with GUI tools;
4) *Simulator* – plugin with classes for the Petri net model simulation;
5) *Util* – plugin with different Renew utility classes.

The plugin also uses the *org.xerial.sqlite-jdbc* library as the SQLite JDBC driver library [11] for implementing the db-net's persistence layer.

The classes which are implemented in the developed db-nets Renew plugin are listed in the Table 1. Only classes implemented/interfaces declared inside plugin are shown together with classes/interfaces which are inherited/implemented by them (the latter are shown without class

members) respectively. The plugin's classes which override only constructor, the GUI and utility classes as well as the autogenerated classes are not listed.

Table 1. The classes implemented in the developed db-nets Renew plugin.

Plugin's class	Extends/Implements	Description of the plugin's class
DBNetsPlugin	de.renew.plugin.PluginAdapter	The DB-Nets Renew tool plugin main class.
DBNetControlLayer	de.renew.net.Net	The db-net's control layer.
ViewPlace	de.renew.net.Place	The db-net's view place.
DBNetTransition	de.renew.net.Transition	The db-net's transition.
ReadArc	de.renew.net.arc.Arc	The db-net's control layer's read arc's constructor.
RollbackArc	de.renew.net.arc.Arc	The db-net's control layer's rollback arc.
DBNetControlLayerInstance	de.renew.net.NetInstanceImpl	The db-net's control layer's instance for the simulation.
ViewPlaceInstance	de.renew.net.MultisetPlaceInstance	The db-net's view place's instance for the simulation.
DBNetTransitionInstance	de.renew.net.TransitionInstance	The db-net's transition's instance for the simulation.
DBNetTransitionOccurrence	de.renew.engine.common.CompositeOccurrence	The db-net's transition's occurrence for the concrete transition firing.
ReadArcOccurence	de.renew.net.arc.ArcOccurrence	The read arc's occurrence for the concrete transition firing.
RollbackArcOccurrence	de.renew.net.arc.ArcOccurrence	The rollback arc's occurrence for the concrete transition firing.

RollbackArcExecutable	de.renew.net.arc.OutputArcExecutable	The rollback arc's executable. Moves the token through the rollback arc from the transition to the place.
SimpleOutputArcExecutable	de.renew.engine.searcher.LateExecutable	The wrapper for the obvious output arc's executable. Allows to NOT move the token through the obvious output arc when the rollback arc is activated.
ReadArcFactory	de.renew.formalism.java.SimpleArcFactory	The factory for the db-net's read arcs' instances creating during the net's parsing and compiling.
RollbackArcFactory	de.renew.formalism.java.SimpleArcFactory	The factory for the db-net's rollback arcs' instances creating during the net's parsing and compiling.
ReadArcBinder	de.renew.net.arc.InputArcBinder	The db-net's read arc's binder. Maps the read arc's variables to the retrieved (from the db-net's persistence layer) values through the corresponding view place's query call.
EditedFact	-	The db-net's data logic layer's action's edited (added/deleted) fact – the rows of the table being added to/deleted from the database

		during the action performing.
Action	-	The db-net's data logic layer's action for modifying the persistence layer's data.
Query	-	The db-net's data logic layer's query for retrieving the persistence layer's data.
ActionCall	de.renew.net.TransitionInscription	The action call – the db-net's data logic layer's action's usage in the concrete db-net's transition.
PreparedStatementWithParams *(the inner class in the ActionCall class)*	-	The edited (added/deleted) fact's prepared statement together with the edited fact's params.
QueryCall	-	The query call – the db-net's data logic layer's query's usage in the concrete db-net's view place.
ActionCallExecutable	de.renew.engine.searcher.LateExecutable	The action call's executable. Executes the transition's action call.
ActionCallValuesBinder	de.renew.engine.searcher.Binder	Binder which maps the variables to the action call's generated values and literals.
JdbcConnection	-	The db-net's persistence layer's JDBC connection data.

DatabaseSchemaDeclaration	-	The db-net's persistence layer's relational database schema declaration.
SingleJavaDBNetCompiler	de.renew.formalism.java.SingleJavaNetCompiler	The db-net's compiler.
SingleJavaDBNetCompilerFactory	implements the de.renew.shadow.ShadowCompilerFactory interface	The factory for creating the db-net's compiler.
DBNetInscriptionParser *(interface)*	de.renew.formalism.java.InscriptionParser	The interface for the db-net's inscription parser.
JavaDBNetParser	de.renew.dbnets.shadow.parser.DBNetInscriptionParser	The JavaCC-based db-net's inscriptions parser.
ParsedDBNetDeclarationNode	de.renew.formalism.java.ParsedDeclarationNode	The parsed db-net's declaration node. Includes the db-net's persistence layer's JDBC connection URL, database schema DDL declarations, data logic layer's queries and actions as well as the net's imports and variables declarations.

The developed db-nets Renew plugin's UML class diagram is shown on the Fig. 7. Only classes implemented/interfaces declared inside plugin are shown together with classes/interfaces which are inherited/implemented by them (the latter are shown without class members) respectively. The plugin's classes which override only constructor, the GUI and utility classes as well as the autogenerated classes are not shown in order to save the visibility.

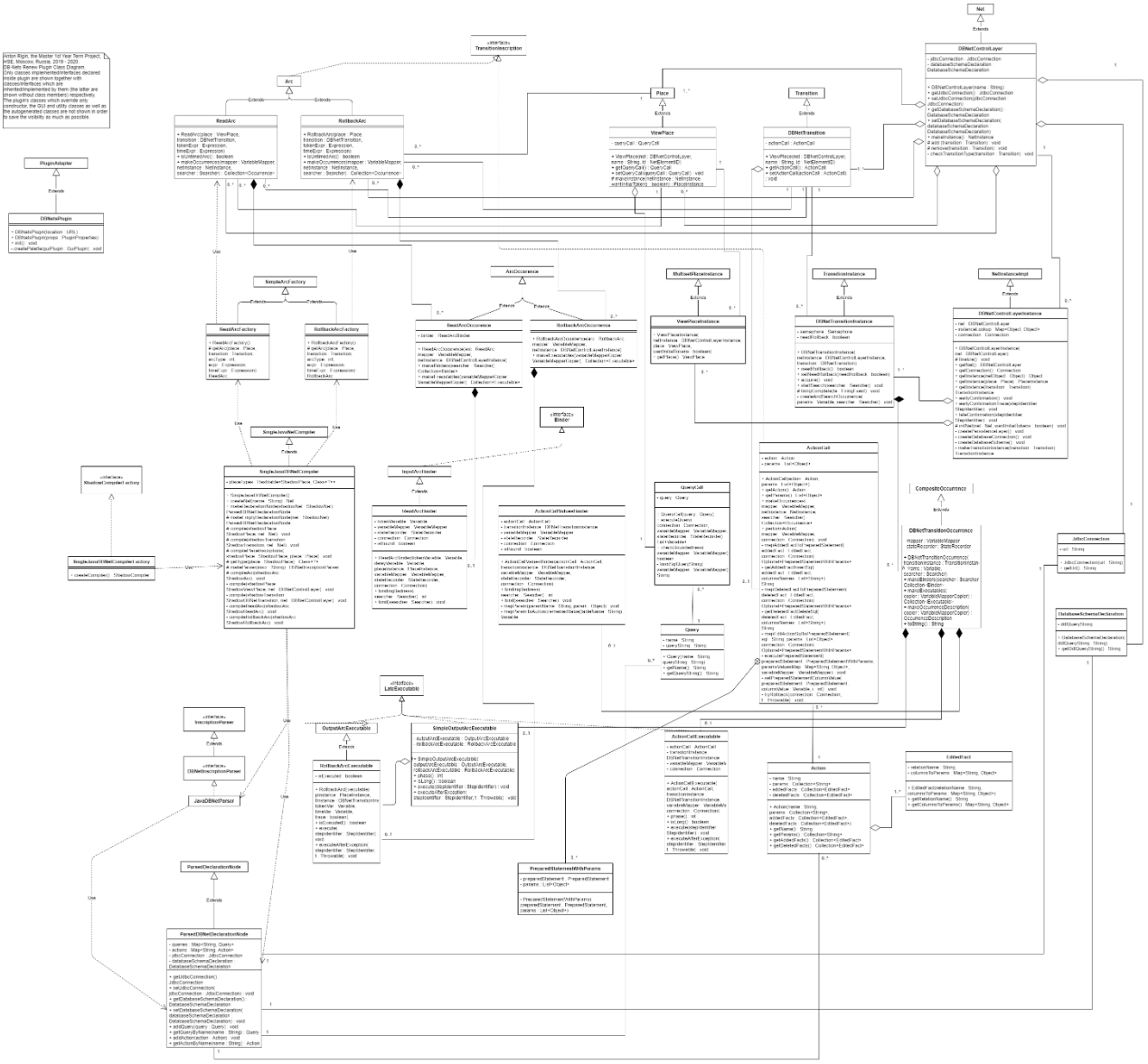

Fig. 7. The developed db-nets Renew plugin's UML class diagram.

For parsing the db-net's text inscriptions (queries, actions and their calls, variables declarations and usages, the db-net's persistence layer's relational database schema and the data consistency rules) the JavaCC-based parser is used which is automatically translated into the *JavaDBNetParser* Java class implementing the db-nets plugin's *de.renew.dbnets.shadow.parser.DBNetInscriptionParser*, which, in its turn, extends the *de.renew.formalism.java.InscriptionParser* interface from the *Formalism* Renew plugin. The parser is based on the Renew's parser for the reference nets implementation.

For implementing the db-net's persistence layer, the SQLite embeddable RDBMS [12] together with the Java Database Connectivity (JDBC) interface is used. The used SQLite JDBC driver library is the *org.xerial.sqlite-jdbc* library [11].

Section 2.3. The DB-net's Control Layer Implementation

As it was mentioned in the section 1.4.3, the db-net's control layer is the modified colored Petri net. The modifications include executing actions in the transitions which update the persistence layer's data, new place type (view places) and two new arc types (read arcs and rollback arcs).

In the current work, the existing functionality of the Renew tool was reused and extended to allow to work with the db-net's control layer's elements. The functionality tied with view places is based on the Renew's places implementation and all the classes tied with the view places extend the corresponding Renew's classes where it is possible. The same is true for the read and rollback arcs and for the db-net transitions. The pieces of the implementation of the colored Petri net's elements behavior which are not overridden by the implementation of the corresponding db-net's elements are executed in the same way as for the corresponding pieces of the implementation of the analogous colored Petri net's elements. The overridden pieces of the implementation are executed based on the db-net's implementation. Since the Renew uses the reference semantics for the tokens, the developed plugin allows to use the data and reference semantics together.

The implementation of the db-net's control layer's elements is as follows.

Section 2.3.1. View Places and Read Arcs Behavior Implementation

View place can be only input place for the transition, and it should be connected with the transition by read arc. View place does not contain any tokens but simulate their existence by retrieving the data from the persistence layer's database using the data logic layer's queries which are bound to them.

In the developed db-nets Renew plugin, the read arc binder, which extends the Renew's input arc binder, executes the related view place's query call (the usage of the query in the concrete view place) on the persistence layer's database. The retrieved result is assigned to the read arc's variables using the fired transition's variables mapper, after which it can be used by the fired transition for performing modifying action and/or moving tokens to other places through the transition's output arcs.

Section 2.3.2. DB-nets' Transitions Behavior Implementation

The transition in the db-net can be bound with the action for modifying the data in the db-net's persistence layer's relational database. The parameters (the values for inserting to and/or

deleting from the database) may give their values in one of the following three ways (depending on the transition's action call declaration during the db-net's designing):

1) from a value of the input arc's variable;
2) from a literal value;
3) as a generated value from the database sequence.

In the first case, the standard input arcs binders as well as the read arcs binders provide all the necessary variables' values. In the second case, the value is assigned to the parameter's variable by the action call values binder based on the parsed literal in the transition's action call declaration. In the third case, the parameter's value in the action call should be string and have the form of the *"dbn_autoincrement_tableName"* (without quotes) where the *tableName* is the name of the persistence layer's database table whose sequence should be used for generating the value. In the last case, the corresponding sequence value is updated using the SQL/DML update query.

Because of the mentioned sequence update, the action call values binder's execution is not guaranteed to be read-only. In conjunction with the concurrent parallel Renew's binders execution it leads to the problem of the race condition including the fact that one generated value may be retrieved several times from the database before its update which, in its turn, may lead to the errors because of the unique constraints violations during performing the actions. To avoid this, the semaphore-based synchronization is used. The action call values binder acquires its transition instance's semaphore permit. As a result, only one thread may execute the action call values binder's code as well as the action performing code (since the sequence update is committed with other transaction's operations in the end of the action performing in order to allow the full rollback in the case if the action performing failed) at one moment. When transition firing ends, the semaphore's permit is released by the transition instance. It allows to eliminate the described problem.

When all the variables are bound with their values, the action is performed through the execution of the prepared statements series – one prepared statement for each action's edited (added/deleted) fact (the database table row). If the error occurs during executing any prepared statement (for example, because of some database schema conditions violation), the transaction is rollbacked and the rollback arc usage is requested.

Section 2.3.3. Rollback Arc Implementation

In the case when the action performing failed (for example, because of some database schema conditions violation), the transaction is rollbacked and the rollback arc's usage is requested instead of the simple output arcs. If the fired transition has a rollback arc, then it will be used as the output arc instead of the simple output arcs. Otherwise, the simple output arcs will still be used.

For handling this, the wrapper was developed for the output arc executable which is responsible for the tokens moving. This wrapper checks if the rollback was requested during the current transition firing. If it was, then the simple output arc executable performing is blocked. In this case, the rollback arc executable will check that the rollback was requested and will execute. Otherwise, the rollback arc will not be used, and the wrapper will allow the output arc to be used as usually.

Section 2.4. The DB-net's Data Logic Layer Implementation

The db-net's data logic layer consists of the queries for retrieving the persistent data and the actions for modifying it as it was described in the section 1.4.2.

Section 2.4.1. Actions Implementation

The actions are declared in the Renew's declaration node for the whole db-net and used in the concrete transition as the action call which also should be declared in the db-net's model. Each action declaration contains the lists of the action's parameters, added facts (inserted database table rows) and deleted facts (deleted database table rows). Each edited (added or deleted) fact consists of the relation (database) table name, names of the columns and their values which should be inserted/deleted. Each such a value may be either the action parameter (in this case it is bound with real value in the start of the corresponding transition firing) or the literal (in this case it is used as declared).

The action is executed as a series of the prepared statements as described in the section 2.3.2.

Section 2.4.2. Queries Implementation

As the actions, the queries are also declared in the Renew's declaration node for the whole db-net and used in the concrete view place as the query call which also should be declared in the db-net's model. The query should be declared as the SQL string using the SQLite dialect. The query may contain the parameters in the form *"${variableName}"* (without quotes) where the *variableName* is the name of variable, whose value should be used for replacing this string. This variable should be known to the fired transition. For example, if the query is *"SELECT id, description FROM ticket WHERE id = ${ticket_id};"* (without quotes) and the *ticket_id* variable's value equals 42, then the *"${ticket_id}"* (without quotes) will be replaced with *42*, and the resulting SQL query will be *"SELECT id, description FROM ticket WHERE id = 42;"*.

After binding all the parameters, the query is executed by the read arc binder as described in the section 2.3.1.

Section 2.5. The DB-net's Persistence Layer Implementation

The persistence layer in the developed db-net's Renew plugin is represented by the SQLite RDBMS. It is connected using the JDBC interface.

For creating the database connection, the JDBC URL is declared in the db-net's declaration node as well as the database schema in the series of the SQL/DDL and DML queries. When the db-net's simulation starts and the db-net's control layer instance is created, it tries to create the connection based on the given JDBC URL. If it is successful, then the SQL/DDL and DML queries are executed to create the necessary database schema. If there are some problems, then simulation is stopped, and the standard Renew tool's error message is outputted for the user.

The created database connection instance is stored in the db-net's control layer's instance and therefore can be accessed by any db-net's control layer's elements instances, occurrences and executables for performing the queries and actions.

When the simulation stops, the database connection is closed either by executing the db-net's control layer's instance's *finalize()* method by the garbage collector or by the next db-net's control layer's instance with the same JDBC URL. For this goal, the static connections map (the connections mapped by their JDBC URLs) is used.

This described technical solution has led to the software product described in the next chapter.

Chapter 3. Developed Software Product

The db-nets Renew plugin is developed. The modelling db-net in the Renew GUI, as well as its simulation, saving and opening are implemented which means that all the functional requirements are met. The SQLite RDBMS serves as the db-net's persistence layer. The Javadoc documentation is provided.

The screenshot of the Renew Petri net model elements palette which includes the options for using the db-net's control layer's elements (four elements in the right bottom corner; from the left to the right – the db-net transition, view place, read arc and rollback arc) is shown on screenshot on the Fig. 8.

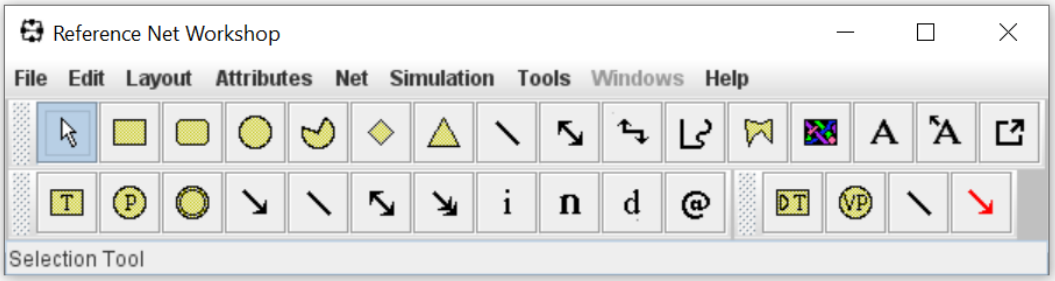

Fig. 8. The screenshot of the Renew Petri net model elements palette with the db-net's control layer's elements.

For the db-net parsing, compiling and simulation, the db-net compiler should be selected in the *Formalisms* item of the *Simulation* menu as shown on the screenshot on the Fig. 9.

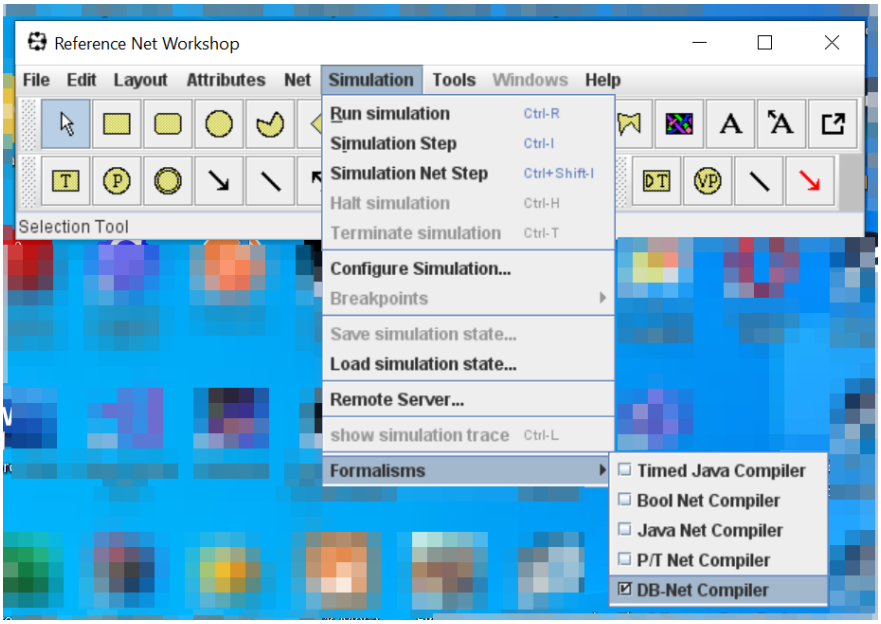

Fig. 9. The db-net compiler selection.

The developed db-nets Renew plugin allows to model (design) the db-net through the standard Renew GUI. The example of the modelled db-net is shown on the screenshots on the Fig. 10 and the Fig. 11. It is the db-net model which was shown in the M. Montali and A. Rivkin's article [6] and was presented in the current paper on the Fig. 5, but now it is without rollback arc since the we will explore the rollback arc behavior further.

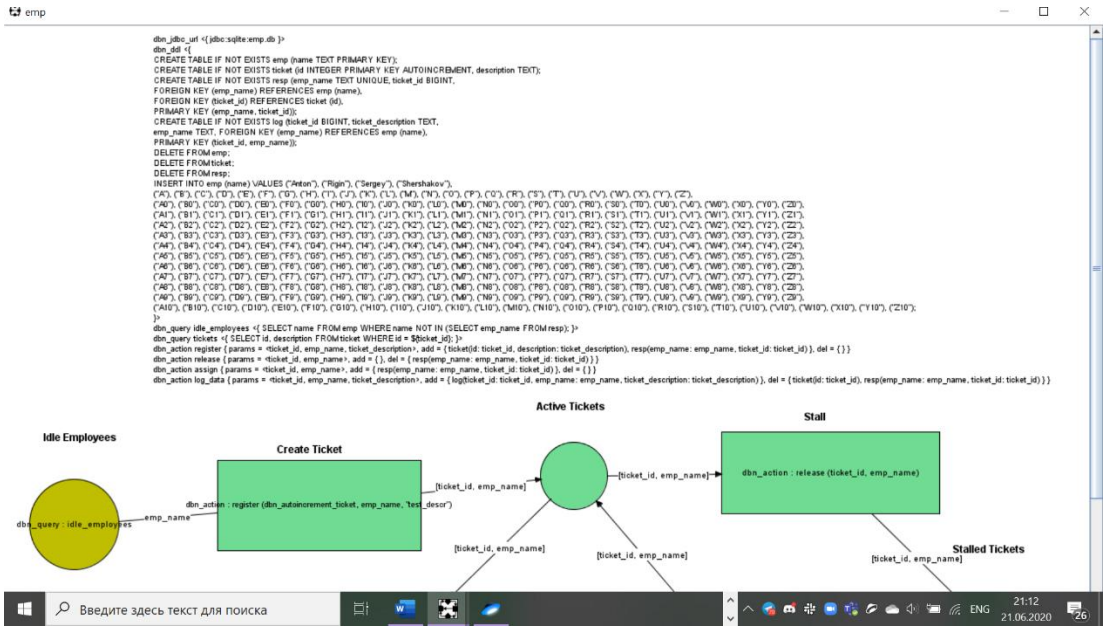

Fig. 10. The example of the db-net for the employees and tickets model (part 1/2).

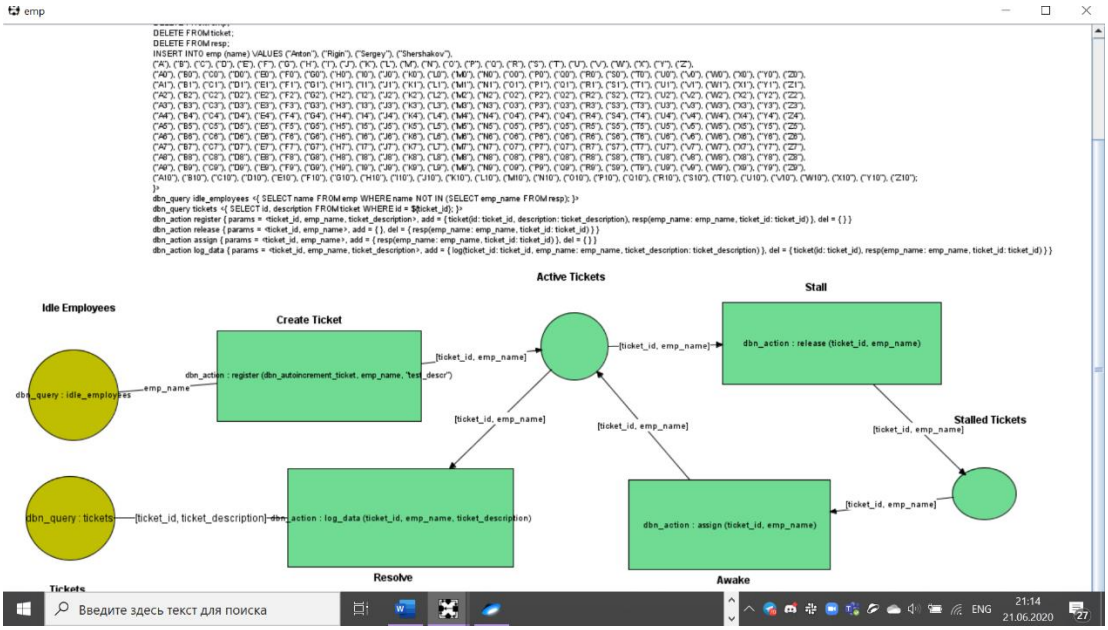

Fig. 11. The example of the db-net for the employees and tickets model (part 2/2).

The multiline text in the top part of the screen is the db-net's declaration node. It contains the JDBC URL declaration for creating the database connection (in this example the *"dbn_jdbc_url <{ jdbc:sqlite:emp.db }>"* string (without quotes) is such a declaration; if such a file does not exist, but it is possible to create it, it will be created automatically), the database schema declaration via the SQL/DDL and DML queries (in the *"dbn_ddl <{ ... }>"* expression), the queries and actions declarations.

In the given example, one of the declared queries is the *tickets* query – *"dbn_query tickets <{ SELECT id, description FROM ticket WHERE id = ${ticket_id}; }>"*. The *"${ticket_id}" expression* is replaced with the *ticket_id* variable's value in the read arc of the view place where the corresponding query call is used. This query retrieves the data from the *tickets* table by the given *id*.

One of the declared actions in the example is the *register* action – *dbn_action register { params = <ticket_id, emp_name, ticket_description>, add = { ticket(id: ticket_id, description: ticket_description), resp(emp_name: emp_name, ticket_id: ticket_id) }, del = { } }*. The action's parameters names here are *ticket_id, emp_name* and *ticket_description*. They are replaced by real values in the transition where the corresponding action call is used. This action adds to the *ticket* relation (database table) the row where the *id* column's value is equal to the *ticket_id* action parameter's value in the corresponding action call and the *description* column's value is equal to the *ticket_description* action parameter's value in the corresponding action call. This row is one of two added facts of this action. Also, the action adds the row into the *resp* table where the *emp_name* column's value is equal to the *emp_name* action parameter's value in the corresponding action call and the *ticket_id* column's value is equal to the *ticket_id* action parameter's value in the corresponding action call. This action does not have deleted facts.

The yellow brown circles (have the *"Idle Employees"* and *"Tickets"* names in the given example) are the view places. The lines starts/ends in them are the read arcs. Other circles are obvious Petri net places and other arcs are obvious Petri net arcs. The rectangles represent the transitions. Each transition should be the db-net transition, the obvious transitions are not allowed in the developed plugin.

Each view place should have the inscription with the query call. In the given example, the *"Idle Employees"* view place have the query call *"dbn_query : idle_employees"* which means that the declared *idle_employees* query is executed to retrieve the persistent data. The *"Tickets"* view place have the query call *"dbn_query : tickets"* which means that the declared *tickets* query is executed to retrieve the data. The aforementioned *"${ticket_id}"* expression in this query will be replaced by the *ticket_id* variable's value.

The transitions may have the action calls, but it is not obligatory. In the given example, all the transitions have the action calls. For example, the *"Create Ticket"* transition have the action call *"dbn_action : register (dbn_autoincrement_ticket, emp_name, "test_descr")"* which means that this transition performs the *register* action when it fires (executes). The *register* action's *ticket_id* parameter is bound with the generated value from the *ticket* table's sequence (because of the *dbn_autoincrement_ticket* parameter value in the action call), the *emp_name* parameter is bound with the *emp_name* variable's value, which is retrieved in the *idle_employees* query call through the *"Idle Employees"* view place and its read arc.

If we will start simulation, we will see the fired transitions in the lighter color as well as the places with the tokens (the number of tokens in the places is shown when it is not 0). This is shown on the screenshot on the Fig. 12.

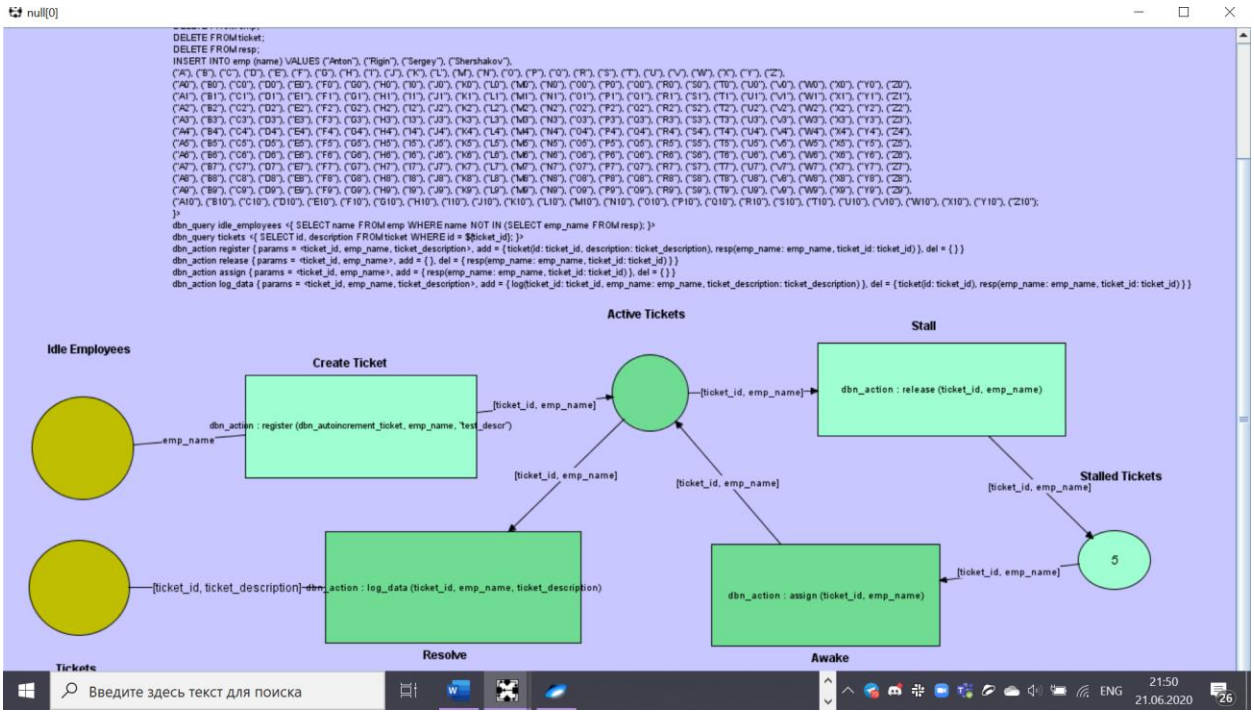

Fig. 12. The simulation running.

We also can present the same model with the rollback arc. It is shown on the screenshot on the Fig. 13. The *assign* action declaration is specially changed in the way to be failed each time starting from the second one because of the unique constraint violation for the *emp_name* column of the *resp* table due to insertion of the same value each time. It allows to catch the rollback faster using step-by-step Renew simulation. The rollback arc on the screenshot has a red color.

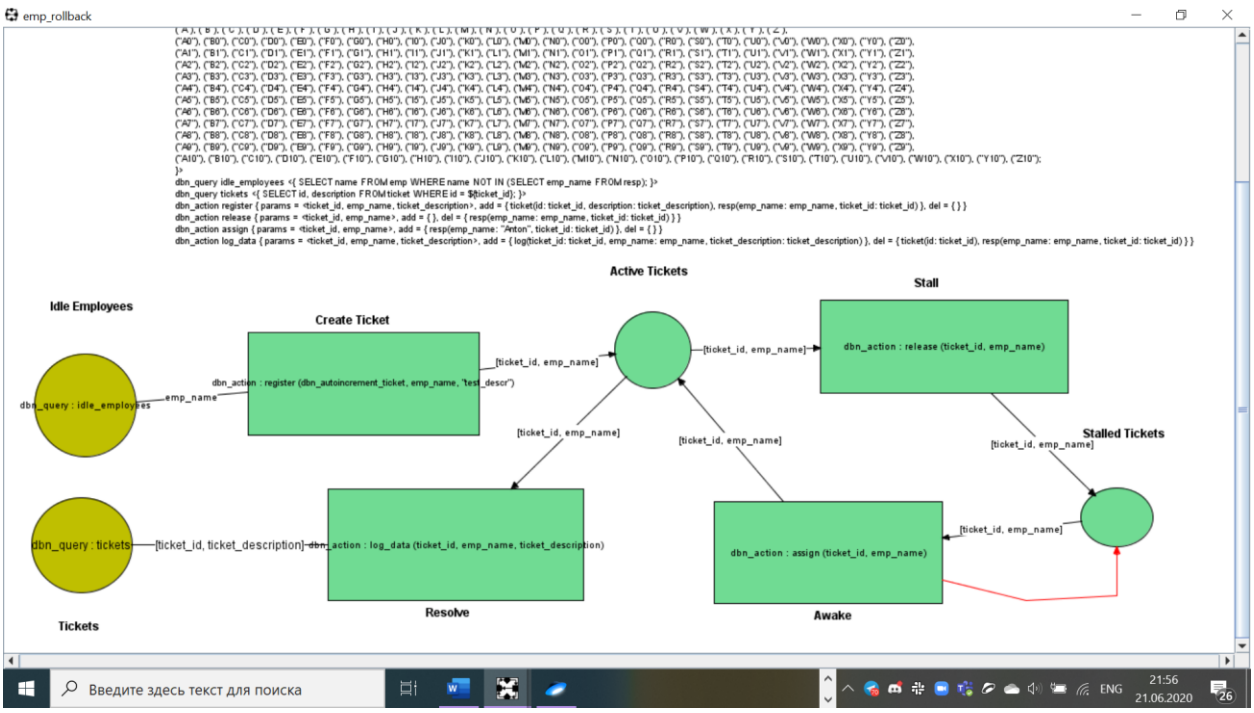

Fig. 13. The model with the rollback arc.

Also, the db-net model for the taxi example from the M. Montali and A. Rivkin's article [6] (it was shown on the Fig. 6 in the current paper) is built in the Renew with the developed db-nets Renew plugin usage. It is shown on the screenshot on the Fig. 14. As it can be seen, not all the db-net's transitions must have the action calls.

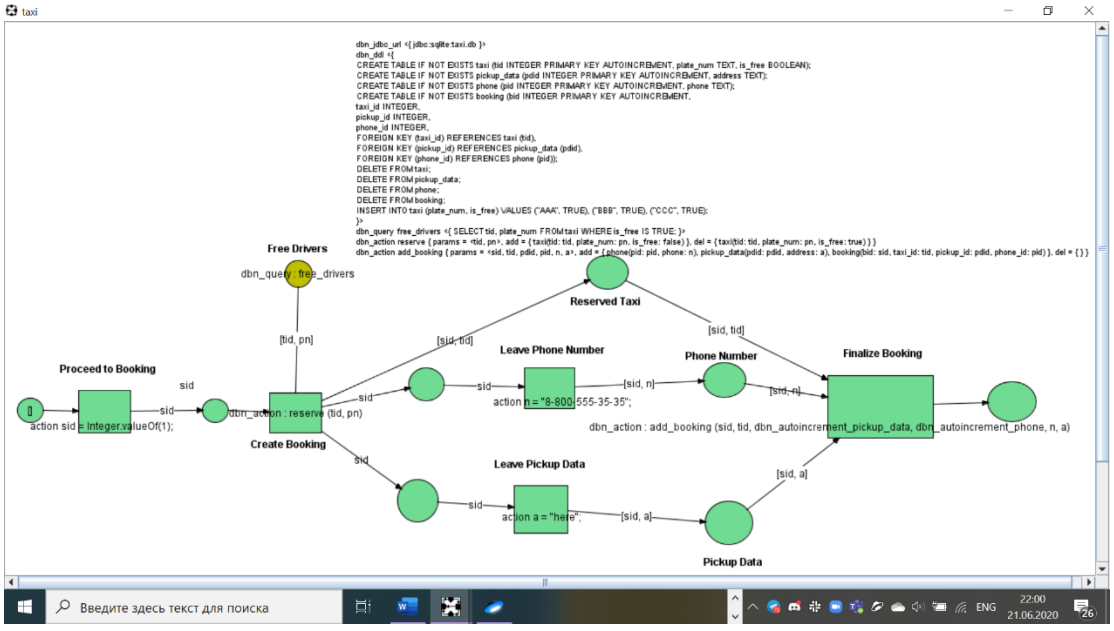

Fig. 14. The db-net model for the taxi example.

If the error during the db-net parsing and/or compiling occurs, it will be outputted in the separate dialog window as it is shown on the screenshot on the Fig. 15.

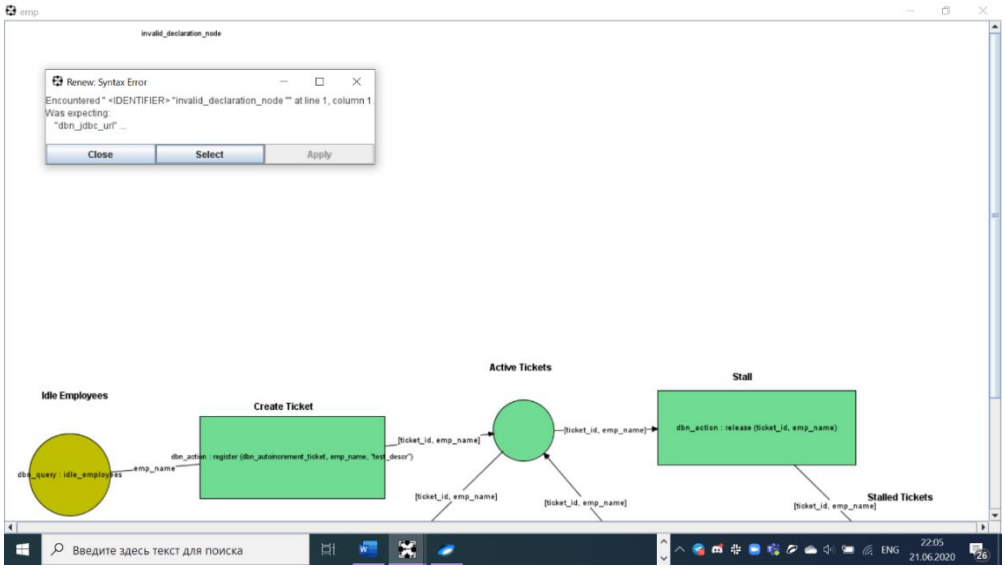

Fig. 15. Parsing/compiling error output example.

Other error messages, as usual, are outputted by Renew in the message line under the palette. The example is shown on the screenshot on the Fig. 16 for the case of the database connection fail due to the invalid JDBC URL.

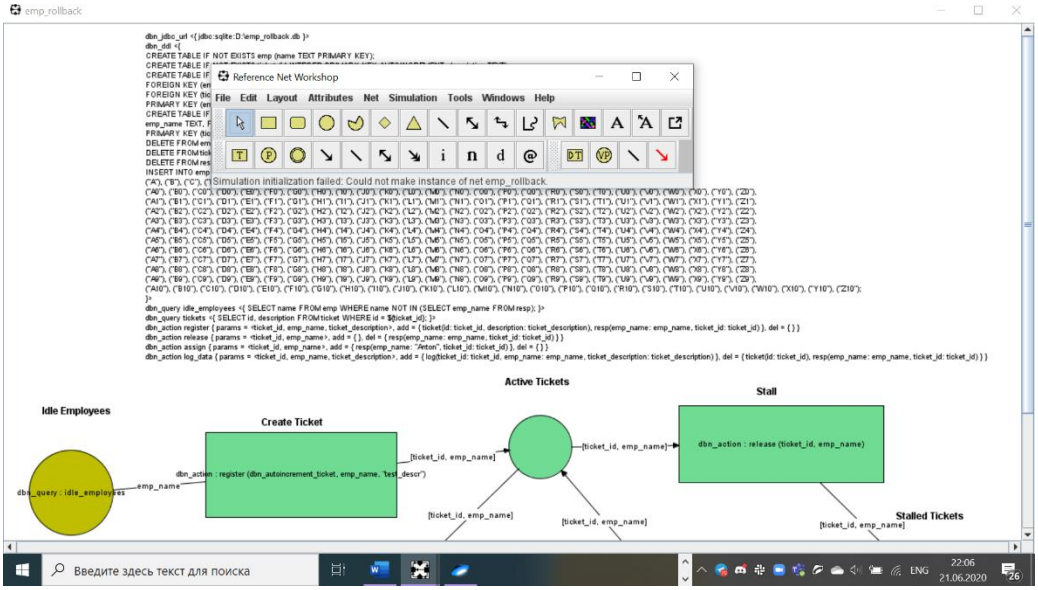

Fig. 16. Database connection error message example.

The developed software product meets the requirements and allows to model the concurrent software systems which use the persistent data. This software product is cross-platform (tested on the Windows 10 and Ubuntu 18.04.4 LTS) since it is written in the Java programming language.

Conclusion

In the current work the program (the software simulator), which supports the db-net formalism, based on the Renew open-source software tool, together with the existing Renew's reference semantics, is developed in the form of the Renew software tool's plugin. The program allows to model the db-nets in the Renew GUI and to simulate them with usage of the SQLite RDBMS in the db-net's persistence layer implementation. The theoretical foundations and the architecture of the simulator are described in the paper.

This program is the first software implementation of the db-net formalism available in public. This simulator can be used in the research tasks as well as in the industrial tasks when the complex concurrent software system which uses the persistent data is under consideration. For instance, the program can be used by researchers including the research staff of the Laboratory of Process-Aware Information Systems at the HSE University's Faculty of Computer Science (PAIS Lab) [8] for conducting the laboratory's researches. Also, this simulator can be used for the further researches and modifying of the db-nets as well as for developing new formalisms which use the persistent data. The work is planned to be presented on a dedicated conference.

Moreover, the simulator is planned to be used in the paper's author's master thesis (in the 2020 – 2021 academic year) which is expected to be the db-net based research of the real software system which is considered by the PAIS Lab. Probably it will be the complex concurrent financial software system since the financial software systems are often considered as safety-critical software systems because of possible extremely large financial losses caused by software failures.

The possible ways of the further development of the plugin are adding ability for user to input the external data during the simulation by the Renew's request, increasing the usability by adding more understandable error outputting and others.

List of References

[1] C.A. Petri and W. Reisig. “Petri net.” Scholarpedia. http://www.scholarpedia.org/article/Petri_net (accessed May 25, 2020).

[2] W. Reisig. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Berlin, Heidelberg, Germany: Springer, 2013.

[3] K. Jensen. “A Brief Introduction to Coloured Petri Nets,” in *Third Int. Workshop on TACAS*, Enschede, the Netherlands, Apr. 2 – 4, 1997, pp. 203 – 208.

[4] K. Jensen and L.M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Berlin, Heidelberg, Germany: Springer, 2009.

[5] “Renew – The Reference Net Workshop.” Renew.de. http://www.renew.de/ (accessed May 25, 2020).

[6] M. Montali and A. Rivkin. “DB-Nets: On the Marriage of Colored Petri Nets and Relational Databases,” in *Transactions on Petri Nets and Other Models of Concurrency XII*, M. Koutny, J. Kleijn, W. Penczek, Eds., Berlin, Heidelberg, Germany: Springer, 2017, pp. 91 – 118.

[7] M. Montali and A. Rivkin. “DB-Nets: On the Marriage of Colored Petri Nets and Relational Databases.” ResearchGate. https://www.researchgate.net/publication/310122815_DB-Nets_on_The_Marriage_of_Colored_Petri_Nets_and_Relational_Databases (accessed May 25, 2020).

[8] “Laboratory of Process-Aware Information Systems (PAIS Lab) — HSE University”. HSE University. https://pais.hse.ru/en/ (accessed May 25, 2020).

[9] B. Farwer. “LLPN – Linear Logic Petri Nets: What are Object Petri Nets?” Wayback Machine. https://web.archive.org/web/20051103131745/http://www.llpn.com/OPNs.html (accessed May 25, 2020).

[10] “File:Forthandback.svg.” Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Forthandback.svg (accessed May 25, 2020).

[11] “Maven Repository: org.xerial >> sqlite-jdbc.” Maven Repository. https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc (accessed May 25, 2020).

[12] “SQLite Home Page.” SQLite.org. https://www.sqlite.org/ (accessed May 25, 2020).

Appendix. The Project's GitHub Source Code Repository

The project's source code is available on the GitHub together with the Renew tool's source code via the following link: https://github.com/Glost/db_nets_renew_plugin, in particular:

- The folder with the documents (including the generated Javadoc files) is available through the following link: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/docs
- The developed plugin's source code is available through the following link: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/prj/sol/projects/renew2.5source/renew2.5/src/DBNets, in particular:
    - The plugin configuration file is available via the following link: https://github.com/Glost/db_nets_renew_plugin/blob/master/root/prj/sol/projects/renew2.5source/renew2.5/src/DBNets/etc/plugin.cfg
    - The developed plugin's Java source files are in the src folder: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/prj/sol/projects/renew2.5source/renew2.5/src/DBNets/src
    - The root package for the most of the plugin's classes is the de.renew.dbnets: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/prj/sol/projects/renew2.5source/renew2.5/src/DBNets/src/de/renew/dbnets
    - The main plugin class is the de.renew.dbnets.DBNetsPlugin: https://github.com/Glost/db_nets_renew_plugin/blob/master/root/prj/sol/projects/renew2.5source/renew2.5/src/DBNets/src/de/renew/dbnets/DBNetsPlugin.java
    - All the Java classes except of some autogenerated are fully commented with the Javadoc comments
- The Renew with the plugin outputs (binaries) are available via the following link: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/prj/sol/output
    - The instructions for their using are available in the corresponding *README.md* file: https://github.com/Glost/db_nets_renew_plugin/blob/master/root/prj/sol/output/README.md
- The developed plugin's used libraries are available through the following link: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/prj/sol/lib
- The samples of the db-nets Renew drawings and the correspondingly created SQLite databases are available via the following link: https://github.com/Glost/db_nets_renew_plugin/tree/master/root/prj/sol/samples