REFERENCE AND DATA SEMANTIC-BASED SIMULATOR OF PETRI NETS EXTENSION WITH THE USE OF RENEW TOOL

Term Project (Coursework)

Student: Anton Rigin   (group mSSE19)

Supervisor: Sergey Shershakov
Senior Lecturer at School of Software Engineering,
Research Fellow at Laboratory of Process-Aware Information Systems
(PAIS Lab), Faculty of Computer Science, HSE

Moscow, 2020

OUTLINE

- Description of subject area
- Basic terms, concepts, definitions
- Relevance and motivation
- Objective and tasks
- Functional requirements
- Renew db-nets plugin architecture & design
- Methods, tools and technologies used
- Results

The project's GitHub repository:
https://github.com/Glost/db_nets_renew_plugin

DESCRIPTION OF SUBJECT AREA

- Concurrent software systems modelling and validation
- Process mining
  - ➢ The techniques of analysis of business processes based on event logs
- Petri nets and their modifications

BASIC TERMS, CONCEPTS, DEFINITIONS (1/8)

- **Petri nets**
  - The powerful tool for modelling the processes and behaviour of concurrent software systems
  - They 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
  - Petri nets support modelling behavior due to **tokens** which form **markings**

BASIC TERMS, CONCEPTS, DEFINITIONS (2/8)

➢ Formally, the Petri net N is the triple $N =$ (P, T, F) [1], where:

    ➢ P is the finite set of places of the net N

    ➢ T is the finite set of transitions of the net N

    ➢ $P \cap T = \emptyset$

    ➢ $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net N

➢ A marking M is a distribution of tokens across places: $M: P \rightarrow \mathbb{N}$
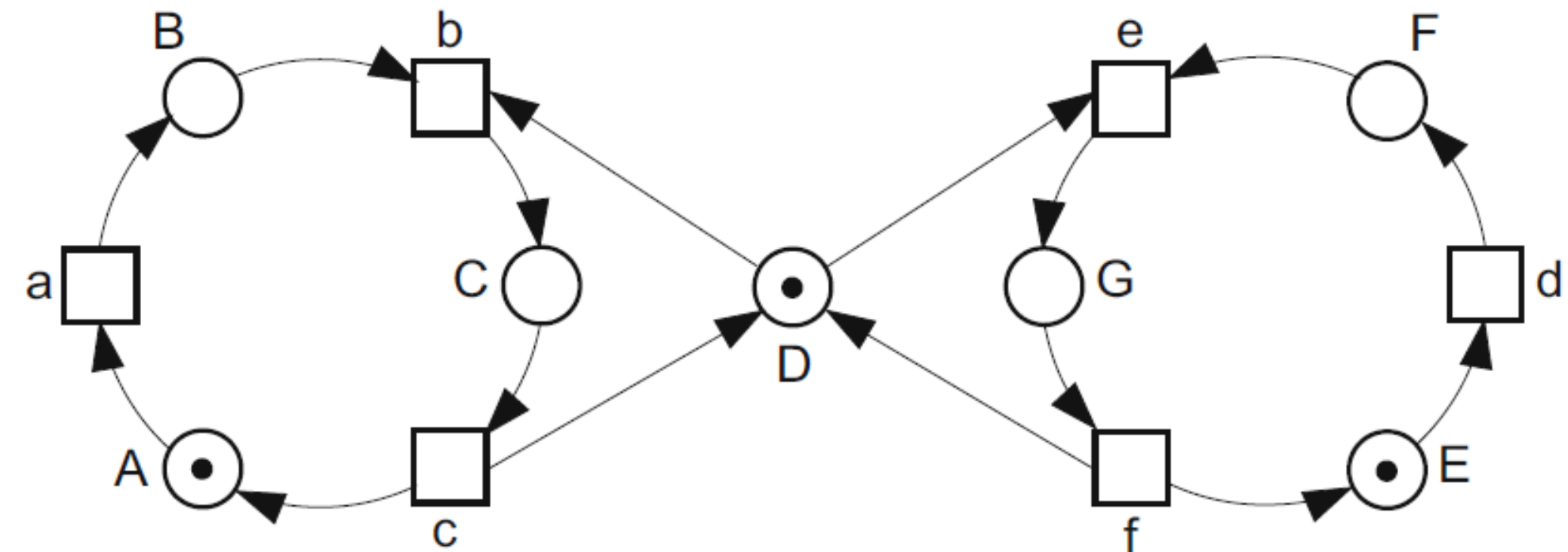
The example of Petri net for modelling the mutual exclusion [1]

BASIC TERMS, CONCEPTS, DEFINITIONS (3/8)

- **Coloured Petri nets**
  - ➢ The Petri nets in which data types of tokens and arc expressions are defined and used [2] [3]
  - ➢ These data types are called **colours** [2] [3]
- **Reference Petri nets**
  - ➢ The Petri nets which have reference semantics rather than value semantics [4]
  - ➢ 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**) [4] [5]

BASIC TERMS, CONCEPTS, DEFINITIONS (4/8)

The example of coloured Petri net for modelling the simple transport protocol [3]
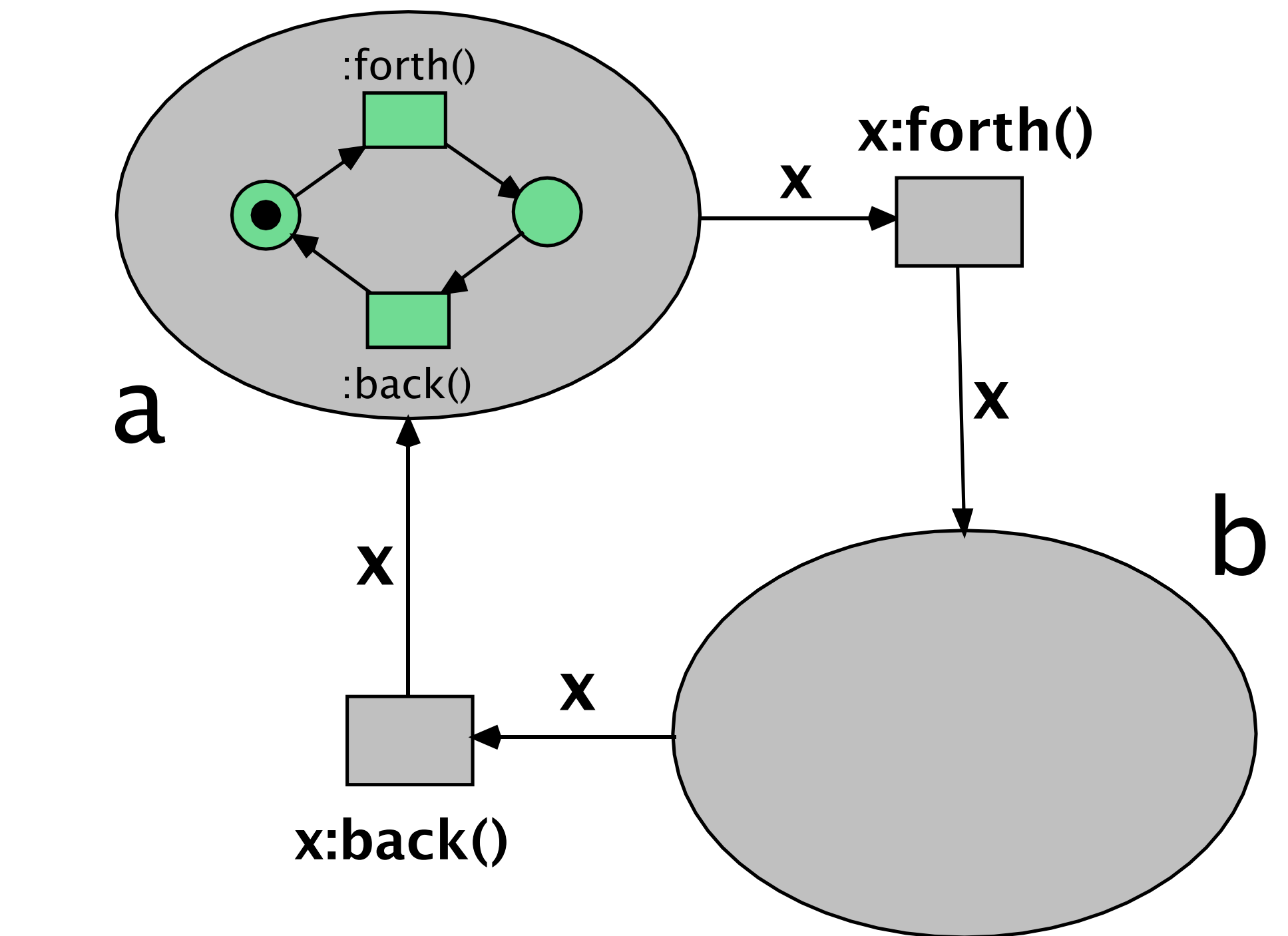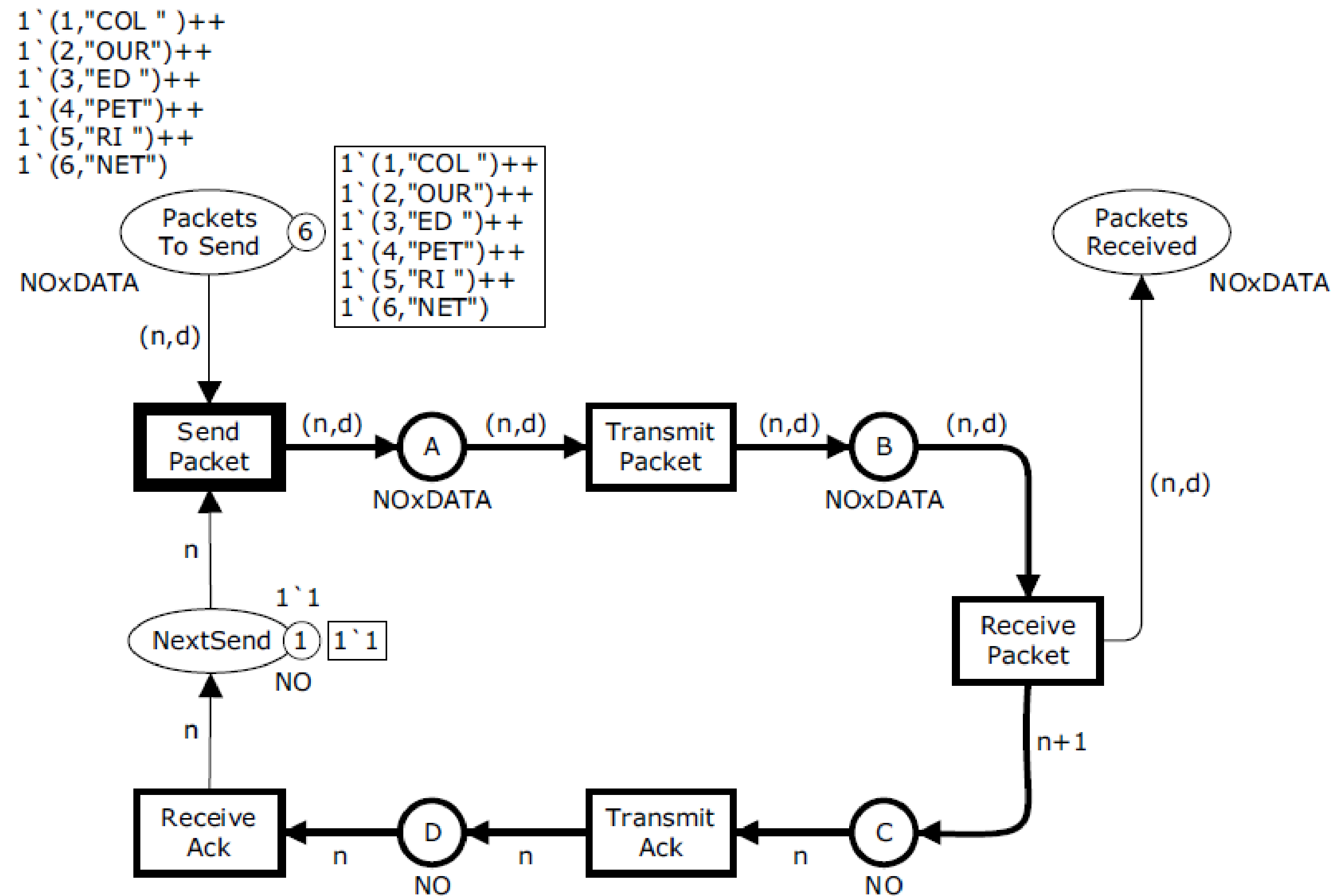
The example of hierarchical reference Petri net [6]

BASIC TERMS, CONCEPTS, DEFINITIONS (5/8)

- **DB-nets**
  - ➢ Were introduced by Marco Montali and Andrey Rivkin in 2016 [7]
  - ➢ DB-net is the model consisting of three layers [7]:
    - ✓ **Control layer** is the modified coloured Petri net
    - ✓ **Data logic layer** interconnects the control layer and persistence layer and allows to access and manipulate the persistent data using **queries** and **actions**
    - ✓ **Persistence layer** represents the relational database with **constraints**
  - ➢ DB-nets allow to consider the process and the persistent data used by this process simultaneously [7], which the other Petri nets modifications cannot
    - ✓ Because of this, it seems to be useful to implement the DB-nets in the software tool since there are no such software implementations available now

BASIC TERMS, CONCEPTS, DEFINITIONS (6/8)
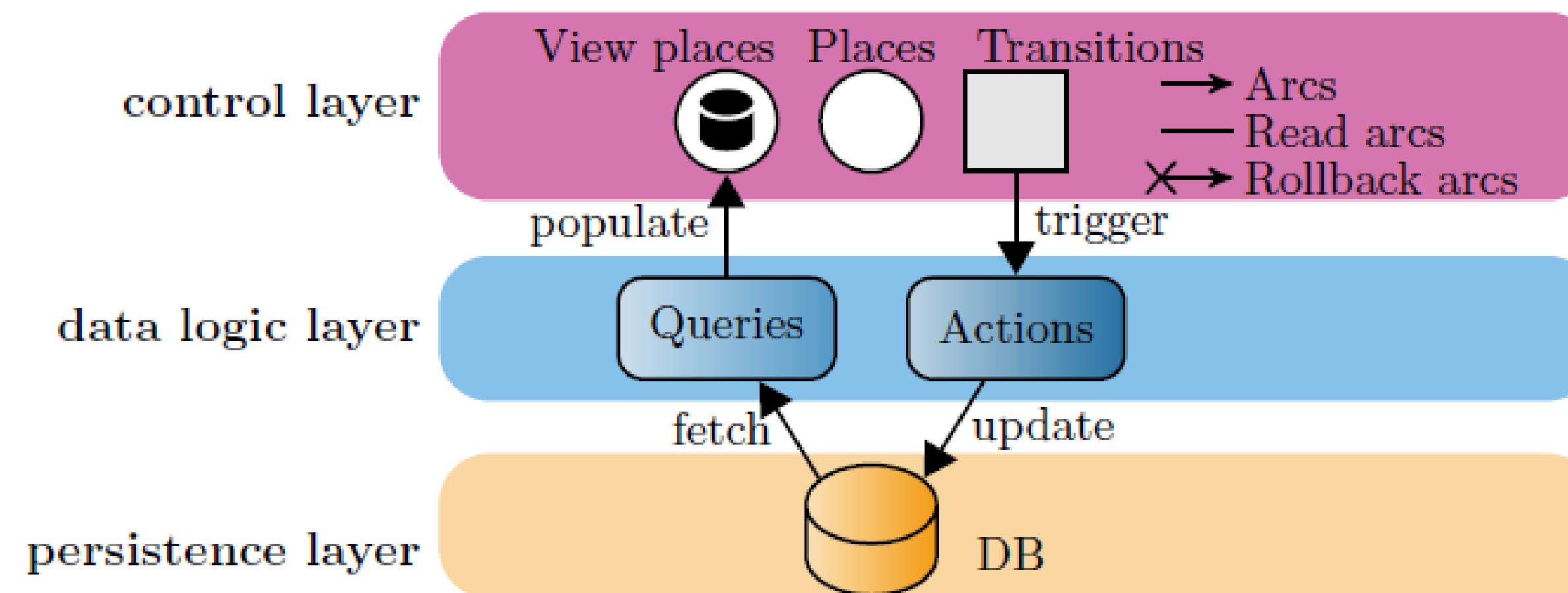
The DB-net structure [8]

BASIC TERMS, CONCEPTS, DEFINITIONS (7/8)
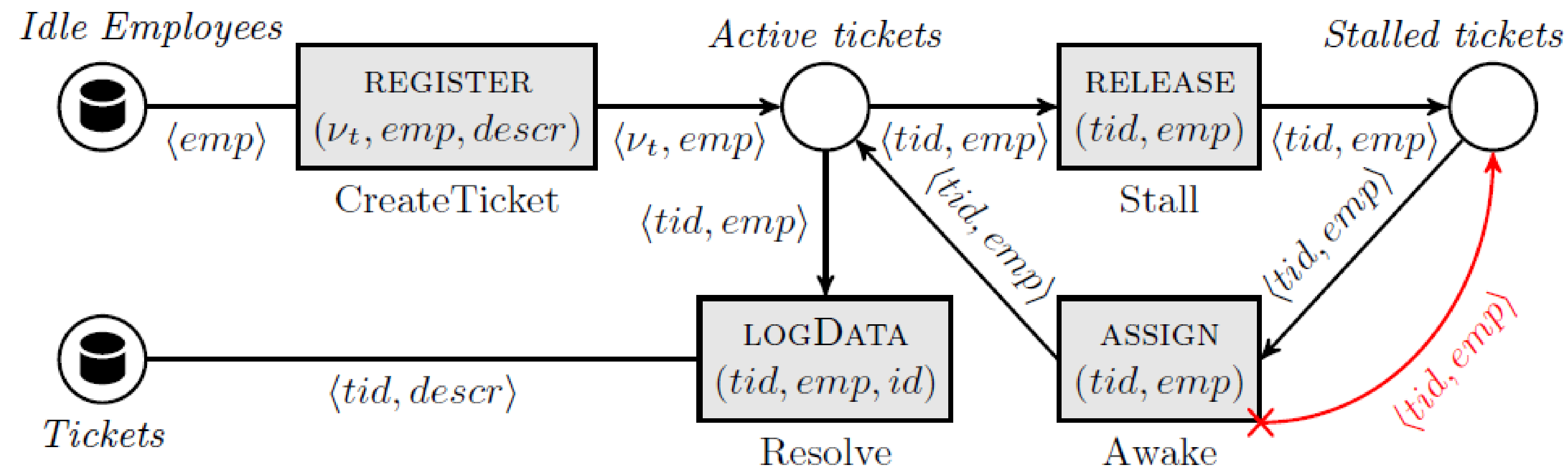
The example of the DB-net's control layer [7]

BASIC TERMS, CONCEPTS, DEFINITIONS (8/8)
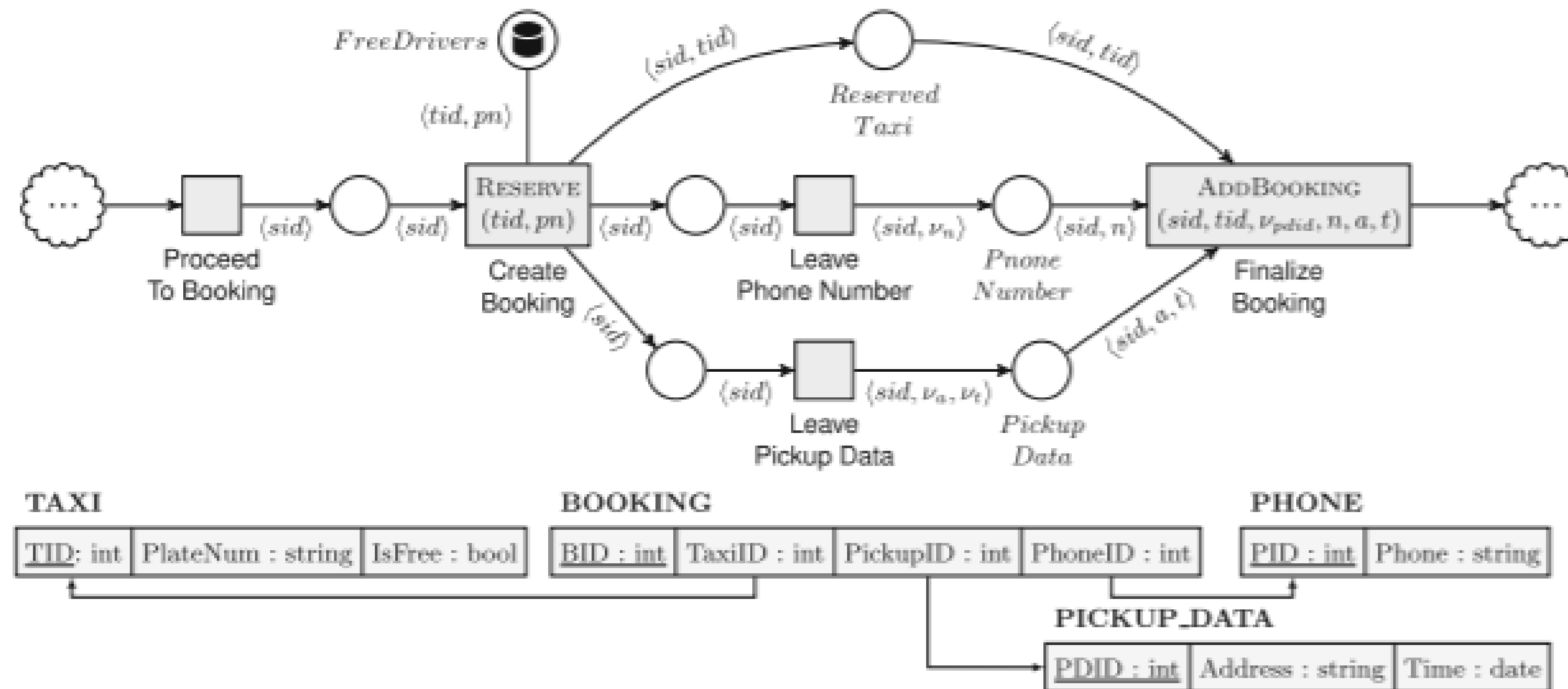
The example of the DB-net [7]

RELEVANCE AND MOTIVATION

- Petri nets are used as a popular technique for modelling the behavior of concurrent programs
- The Petri nets and their modifications implemented in many different software tools (**CPN Tools** for the coloured Petri nets [9], **Renew** for the reference Petri nets [5] and so on)
  - ➢ However, the standard Petri nets as well as coloured Petri nets and reference Petri nets cannot consider the state of persistent data
- As it was already mentioned, the DB-nets solve this problem [7]
  - ➢ However, there is no published program implementations of the DB-nets
- **Renew** is a powerful software tool which implements reference Petri nets [5]
- Therefore, it should be useful to implement the DB-nets with the reference semantics support as the plugin for the **Renew**
- This implementation allows to model the behavior of the programs with reference semantics (which are very common in such programming languages as Java) with considering the state of persistent data

OBJECTIVE AND TASKS

- **Objective**
  - ➤ Implementing the software tool for simulating DB-nets with reference semantics support as the plugin for the Renew
- **Tasks**
  - ➤ Learning the existing Renew's design and code
  - ➤ Formulating the formal functional requirements for the software tool (the DB-nets Renew plugin)
  - ➤ Developing the design of the software tool
  - ➤ Implementing the DB-nets concept in the form of Renew plugin
  - ➤ Testing the developed software tool using some real enterprise data or artificially generated data
  - ➤ Documenting the developed software tool

FUNCTIONAL REQUIREMENTS (1/2)

- 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
- The program shall allow user to create the database schema at the db-net persistence layer using the Renew tool's GUI
- The program shall allow user to declare queries and actions for the db-net data logic layer using the Renew tool's GUI

FUNCTIONAL REQUIREMENTS (2/2)

- The program shall allow user to simulate the modeled db-net's run using the Renew tool's GUI
- The program shall allow user to save the modeled db-net using the Renew tool's GUI
- The program shall allow user to open the previously saved modeled db-net using the Renew tool's GUI

RENEW DB-NETS PLUGIN ARCHITECTURE & DESIGN (1/4)

- Renew is plugin-based – each piece of code, each class belongs to one of its plugins [5]
- Renew is written in Java as well as its plugins [5]
- Renew uses Apache Ant as a build system [5]
- The DB-net plugin is developed using the existing Renew base plugins as examples
- The DB-net plugin extends the classes of the following base Renew plugins:
  - ➢ CH (DrawPlugin) – plugin with number of drawing utils
  - ➢ Formalism – plugin with number of Petri nets' and their elements' formalisms
  - ➢ GUI – plugin with number of GUI tools
  - ➢ Simulator – plugin for simulating the modelled Petri net
  - ➢ Util – plugin with number of different Renew utility classes
- The plugin also uses the org.xerial.sqlite-jdbc library as the SQLite JDBC driver library [10] for implementing the db-net's persistence layer.

RENEW DB-NETS PLUGIN ARCHITECTURE & DESIGN (2/4)

The Renew DB-Nets Plugin Class Diagram – on the next slides

Net

DBNetControlLayer
- jdbcConnection : JdbcConnection
- databaseSchemaDeclaration : DatabaseSchemaDeclaration

+ DBNetControlLayer(name : String)
+ getJdbcConnection() : JdbcConnection
+ setJdbcConnection(jdbcConnection : JdbcConnection)
+ getDatabaseSchemaDeclaration() : DatabaseSchemaDeclaration
+ setDatabaseSchemaDeclaration(databaseSchemaDeclaration : DatabaseSchemaDeclaration)
+ makeInstance() : NetInstance
# add(transition : Transition) : void
# remove(transition : Transition) : void
- checkTransitionType(transition : Transition) : void

PluginAdapter

DBNetsPlugin
+ DBNetsPlugin(location : URL)
+ DBNetsPlugin(props : PluginProperties)
+ init() : void
- createPalette(guiPlugin : GuiPlugin) : void

«interface» TransitionInscription

Arc

ReadArc
+ ReadArc(place : ViewPlace, transition : DBNetTransition, tokenExpr : Expression, timeExpr : Expression)
+ isUntimedArc() : boolean
+ makeOccurrences(mapper : VariableMapper, netInstance : NetInstance, searcher : Searcher) : Collection<Occurrence>

RollbackArc
+ RollbackArc(place : Place, transition : DBNetTransition, tokenExpr : Expression, timeExpr : Expression)
+ isUntimedArc() : boolean
+ makeOccurrences(mapper : VariableMapper, netInstance : NetInstance, searcher : Searcher) : Collection<Occurrence>

Place

ViewPlace
- queryCall : QueryCall

+ ViewPlace(net : DBNetControlLayer, name : String, id : NetElementID)
+ getQueryCall() : QueryCall
+ setQueryCall(queryCall : QueryCall) : void
# makeInstance(netInstance : NetInstance, wantInitialTokens : boolean) : PlaceInstance

Transition

DBNetTransition
- actionCall : ActionCall

+ DBNetTransition(net : DBNetControlLayer, name : String, id : NetElementID)
+ getActionCall() : ActionCall
+ setActionCall(actionCall : ActionCall) : void

SimpleArcFactory

ArcOccurence

MultisetPlaceInstance

TransitionInstance

NetInstanceImpl

ReadArcFactory
+ ReadArcFactory()
# getArc(place : Place, transition : Transition, arcType : int, expr : Expression, timeExpr : Expression) : ReadArc

RollbackArcFactory
+ RollbackArcFactory()
# getArc(place : Place, transition : Transition, arcType : int, expr : Expression, timeExpr : Expression) : RollbackArc

ReadArcOccurence
- binder : ReadArcBinder

+ ReadArcOccurence(arc : ReadArc, mapper : VariableMapper, netInstance : DBNetControlLayerInstance)
+ makeBinders(searcher : Searcher) : Collection<Binder>
+ makeExecutables(variableMapperCopier : VariableMapperCopier) : Collection<Executable>

RollbackArcOccurence
+ RollbackArcOccurence(arc : RollbackArc, mapper : VariableMapper, netInstance : DBNetControlLayerInstance)
+ makeExecutables(variableMapperCopier : VariableMapperCopier) : Collection<Executable>

ViewPlaceInstance
+ ViewPlaceInstance(net : DBNetControlLayerInstance, place : ViewPlace, wantInitialTokens : boolean)
+ getPlace() : ViewPlace

DBNetTransitionInstance
- semaphore : Semaphore
- needRollback : boolean

+ DBNetTransitionInstance(netInstance : DBNetControlLayerInstance, transition : DBNetTransition)
+ needRollback() : boolean
+ setNeedRollback(needRollback : boolean) : void
+ acquire() : void
+ startSearch(searcher : Searcher) : void
# firingComplete(fe : FiringEvent) : void
- createAndSearchOccurrence(params : Variable, searcher : Searcher) : void

DBNetControlLayerInstance
- net : DBNetControlLayer
- instanceLookup : Map<Object, Object>
- connection : Connection

- DBNetControlLayerInstance(net : DBNetControlLayer)
finalize() : void
+ getNet() : DBNetControlLayer
+ getConnection() : Connection
+ getInstance(netObject : Object) : Object
+ getInstance(place : Place) : PlaceInstance
+ getInstance(transition : Transition) : TransitionInstance
+ earlyConfirmation() : void
+ earlyConfirmationTrace(stepIdentifier : StepIdentifier) : void
+ lateConfirmation(stepIdentifier : StepIdentifier) : void
# init(net : Net, wantInitialTokens : boolean) : void
- createPersistenceLayer() : void
- createDatabaseConnection() : void
- createDatabaseSchema() : void
- makeTransitionInstance(transition : Transition) : TransitionInstance

SingleJavaNetCompiler

«interface» ShadowCompilerFactory

SingleJavaDBNetCompiler
- placeTypes : Hashtable<ShadowPlace, Class<?>>

+ SingleJavaDBNetCompiler()
+ createNet(name : String) : Net
- makeDeclarationNode(shadowNet : ShadowNet) : ParsedDBNetDeclarationNode
- makeEmptyDeclarationNode(net : ShadowNet) : ParsedDBNetDeclarationNode
- compile(shadowPlace : ShadowPlace, net : Net) : void
- compile(shadowTransition : ShadowTransition, net : Net) : void
- compilePlaceInscriptions(shadowPlace : ShadowPlace, place : Place) : void
getType(place : ShadowPlace) : Class<?>
- makeParser(inscr : String) : DBNetInscriptionParser
- compile(shadowArc : ShadowArc, net : DBNetControlLayer) : void
- compile(shadowViewPlace, net : DBNetControlLayer) : void
- compile(shadowDBNetTransition, net : DBNetControlLayer) : void
- compileReadArc(shadowArc : ShadowReadArc) : void
- compileRollbackArc(shadowRollbackArc) : void

SingleJavaDBNetCompilerFactory
+ createCompiler() : ShadowCompiler

InputArcBinder

«interface» Binder

ReadArcBinder
- tokenVariable : Variable
- variableMapper : VariableMapper
- stateRecorder : StateRecorder
- connection : Connection
- isBound : boolean

+ ReadArcBinder(tokenVariable : Variable, delayVariable : Variable, placeInstance : PlaceInstance, variableMapper : VariableMapper, stateRecorder : StateRecorder, connection : Connection)
+ bindingBadness() : int
+ bind(searcher : Searcher) : void
+ mapParam(paramName : String, param : Object) : void
+ mapParamToAutoincrementedValue(tableName : String) : Variable

ActionCallValuesBinder
- actionCall : ActionCall
- transitionInstance : DBNetTransitionInstance
- variableMapper : VariableMapper
- stateRecorder : StateRecorder
- connection : Connection
- isBound : boolean

+ ActionCallValuesBinder(actionCall : ActionCall, transitionInstance : DBNetTransitionInstance, variableMapper : VariableMapper, stateRecorder : StateRecorder, connection : Connection)
+ bindingBadness() : int
+ bind(searcher : Searcher) : void

QueryCall
- query : Query

+ QueryCall(query : Query)
+ executeQuery(connection : Connection, variableMapper : VariableMapper, stateRecorder : StateRecorder) : List<Variable>
+ checkBoundedness(variableMapper : VariableMapper) : boolean
+ formSqlQueryString(variableMapper : VariableMapper) : String

ActionCall
- action : Action
- params : List<Object>

+ ActionCall(action : Action, params : List<Object>)
+ getAction() : Action
+ getParams() : List<Object>
+ makeOccurrences(mapper : VariableMapper, netInstance : NetInstance, searcher : Searcher) : Collection<Occurrence>
+ performAction(variableMapper : VariableMapper, connection : Connection) : void
- mapAddedFactToPreparedStatement(addedFact : EditedFact, connection : Connection) : Optional<PreparedStatementWithParams>
- getAddedFactInsertSql(addedFact : EditedFact, columnNames : List<String>) : String
- mapDeletedFactToPreparedStatement(deletedFact : EditedFact, connection : Connection) : Optional<PreparedStatementWithParams>
- getDeletedFactDeleteSql(deletedFact : EditedFact, columnNames : List<String>) : String
- mapEditActionSqlToPreparedStatement(sql : String, params : List<Object>, connection : Connection) : Optional<PreparedStatementWithParams>
- executePreparedStatement(preparedStatement : PreparedStatementWithParams, paramsValuesMap : Map<String, Object>, variableMapper : VariableMapper) : void
- setPreparedStatementColumnValue(preparedStatement : PreparedStatement, columnValue : Variable, i : int) : void
- tryRollback(connection : Connection, t : Throwable) : void

CompositeOccurence

DBNetTransitionOccurrence
- mapper : VariableMapper
- stateRecorder : StateRecorder

+ DBNetTransitionOccurrence(transitionInstance : DBNetTransitionInstance, trans : Variable, searcher : Searcher)
+ makeBinders(searcher : Searcher) : Collection<Binder>
+ makeExecutables(copier : VariableMapperCopier) : Collection<Executable>
+ makeOccurrenceDescription(copier : VariableMapperCopier) : OccurrenceDescription
+ toString() : String

JdbcConnection
- url : String

+ JdbcConnection(url : String)
+ getUrl() : String

DatabaseSchemaDeclaration
- ddlQueryString

+ DatabaseSchemaDeclaration(ddlQueryString : String)
+ getDdlQueryString() : String

«interface» InscriptionParser

«interface» DBNetInscriptionParser

JavaDBNetParser

«interface» LateExecutable

OutputArcExecutable

SimpleOutputArcExecutable
- outputArcExecutable : OutputArcExecutable
- rollbackArcExecutable : RollbackArcExecutable

+ SimpleOutputArcExecutable(outputArcExecutable : OutputArcExecutable, rollbackArcExecutable : RollbackArcExecutable)
+ phase() : int
+ isLong() : boolean
+ execute(stepIdentifier : StepIdentifier) : void
+ executeAfterException(stepIdentifier : StepIdentifier, t : Throwable) : void

RollbackArcExecutable
- isExecuted : boolean

+ RollbackArcExecutable(pInstance : PlaceInstance, tInstance : DBNetTransitionInstance, tokenVar : Variable, timeVar : Variable, trace : boolean)
+ isExecuted() : boolean
+ execute(stepIdentifier : StepIdentifier) : void
+ executeAfterException(stepIdentifier : StepIdentifier, t : Throwable) : void

Query
- name : String
- queryString : String

+ Query(name : String, queryString : String)
+ getName() : String
+ getQueryString() : String

ActionCallExecutable
- actionCall : ActionCall
- transitionInstance : DBNetTransitionInstance
- variableMapper : VariableMapper
- connection : Connection

+ ActionCallExecutable(actionCall : ActionCall, transitionInstance : DBNetTransitionInstance, variableMapper : VariableMapper, connection : Connection)
+ phase() : int
+ isLong() : boolean
+ execute(stepIdentifier : StepIdentifier) : void
+ executeAfterException(stepIdentifier : StepIdentifier, t : Throwable) : void

Action
- name : String
- params : Collection<String>
- addedFacts : Collection<EditedFact>
- deletedFacts : Collection<EditedFact>

+ Action(name : String, params : Collection<String>, addedFacts : Collection<EditedFact>, deletedFacts : Collection<EditedFact>)
+ getName() : String
+ getParams() : Collection<String>
+ getAddedFacts() : Collection<EditedFact>
+ getDeletedFacts() : Collection<EditedFact>

EditedFact
- relationName : String
- columnsToParams : Map<String, Object>

+ EditedFact(relationName : String, columnsToParams : Map<String, Object>)
+ getRelationName() : String
+ getColumnsToParams() : Map<String, Object>

PreparedStatementWithParams
- preparedStatement : PreparedStatement
- params : List<Object>

- PreparedStatementWithParams(preparedStatement : PreparedStatement, params : List<Object>)

ParsedDeclarationNode

ParsedDBNetDeclarationNode
- queries : Map<String, Query>
- actions : Map<String, Action>
- jdbcConnection : JdbcConnection
- databaseSchemaDeclaration : DatabaseSchemaDeclaration

+ getJdbcConnection() : JdbcConnection
+ setJdbcConnection(jdbcConnection : JdbcConnection) : void
+ getDatabaseSchemaDeclaration() : DatabaseSchemaDeclaration
+ setDatabaseSchemaDeclaration(databaseSchemaDeclaration : DatabaseSchemaDeclaration) : void
+ addQuery(query : Query) : void
+ getQueryByName(name : String) : Query
+ addAction(action : Action) : void
+ getActionByName(name : String) : Action

Net

Extends

DBNetControlLayer

- jdbcConnection : JdbcConnection
- databaseSchemaDeclaration : DatabaseSchemaDeclaration

+ DBNetControlLayer(name : String)
+ getJdbcConnection() : JdbcConnection
+ setJdbcConnection(jdbcConnection : JdbcConnection)
+ getDatabaseSchemaDeclaration() : DatabaseSchemaDeclaration
+ setDatabaseSchemaDeclaration(databaseSchemaDeclaration : DatabaseSchemaDeclaration)
+ makeInstance() : NetInstance
# add (transition : Transition) : void
# remove(transition : Transition) : void
- checkTransitionType(transition : Transition) : void

1

Place

1..*

1

Extends

ViewPlace

- queryCall : QueryCall

+ ViewPlace(net : DBNetControlLayer, name : String, id : NetElementID)
+ getQueryCall() : QueryCall
+ setQueryCall(queryCall : QueryCall) : void
# makeInstance(netInstance : NetInstance, wantInitialTokens : boolean) : PlaceInstance

Transition

Extends

DBNetTransition

- actionCall : ActionCall

+ ViewPlace(net : DBNetControlLayer, name : String, id : NetElementID)
+ getActionCall() : ActionCall
+ setActionCall(actionCall : ActionCall) : void

1..*

1

1   1   1

1   0..*

1

MultisetPlaceInstance

Extends

0..*

TransitionInstance

Extends

NetInstanceImpl

Extends

0..*

...ArcOccurrence

...nce(arc : RollbackArc,
...per,
...ControlLayerInstance)
...VariableMapperCopier :
...r) : Collection<Executable>

0..*

ViewPlaceInstance

+ ViewPlaceInstance(
netInstance : DBNetControlLayerInstance,
place : ViewPlace,
wantInitialTokens : boolean)
+ getPlace() : ViewPlace

0..*

DBNetTransitionInstance

- semaphore : Semaphore
- needRollback : boolean

+ DBNetTransitionInstance(
netInstance : DBNetControlLayerInstance,
transition : DBNetTransition)
+ needRollback() : boolean
+ setNeedRollback(needRollback : boolean)
+ acquire() : void
+ startSearch(searcher : Searcher) : void
# firingComplete(fe : FiringEvent) : void
- createAndSearchOccurrence(
params : Variable, searcher : Searcher) : void

1..*

DBNetControlLayerInstance

- net : DBNetControlLayer
- instanceLookup : Map<Object, Object>
- connection : Connection

+ DBNetControlLayerInstance(
net : DBNetControlLayer)
finalize() : void
+ getNet() : DBNetControlLayer
+ getConnection() : Connection
+ getInstance(netObject : Object) : Object
+ getInstance(place : Place) : PlaceInstance
+ getInstance(transition : Transition) : TransitionInstance
+ earlyConfirmation() : void
+ earlyConfirmationTrace(stepIdentifier : StepIdentifier) : void
+ lateConfirmation(stepIdentifier : StepIdentifier) : void
# initNet(net : Net, wantInitialTokens : boolean) : void
- createPersistenceLayer() : void
- createDatabaseConnection() : void
- createDatabaseSchema() : void
- makeTransitionInstance(transition : Transition) : TransitionInstance

0..*

ActionCall

- action : Action
- params : List<Object>

+ ActionCall(action : Action,
params : List<Object>)

METHODS, TOOLS AND TECHNOLOGIES USED

- The DB-net with the reference semantics support is a core model used in the project
- The methods of constructing DB-net and simulating its run are used
- Renew is a main software tool used in the project
- Technological stack: Java SE 8, Apache Ant, JavaCC and SQLite (the latter is used as the relational DBMS for the persistent layer of the DB-net)
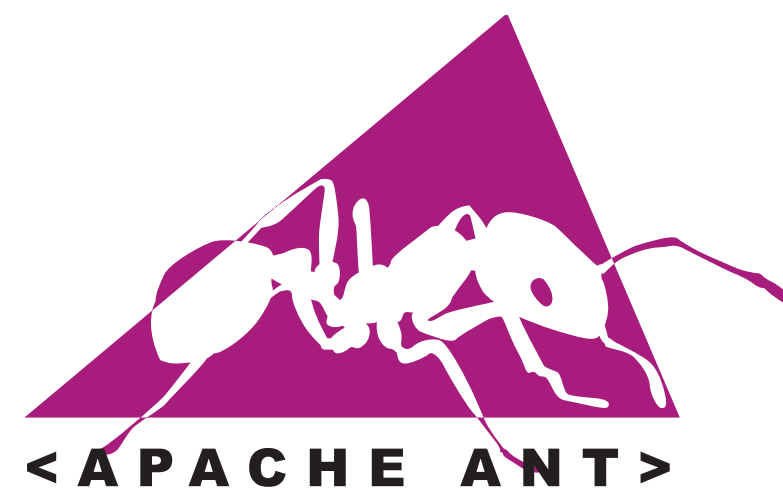
RESULTS (1/4)
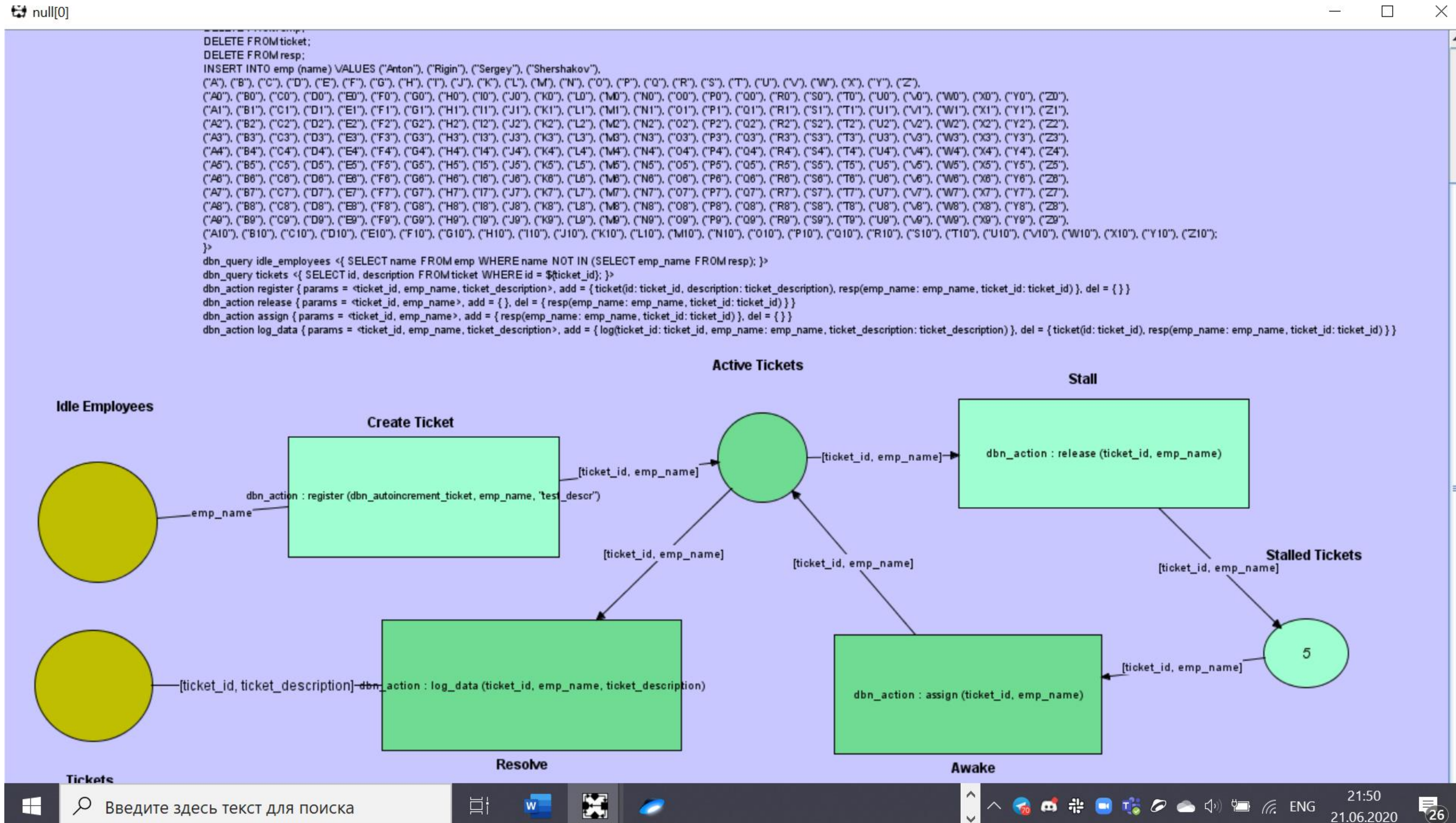
RESULTS (2/4)

- The developed software tool 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 developed software tool is applicable for cases where it is necessary to consider both process and the persistent data
- This program is the first software implementation of the db-net formalism available in public

RESULTS (3/4)

The demonstration

RESULTS (4/4)

- The term project can be continued in the 2nd year as the master thesis which will possibly include development of some new features for the developed software tool and conducting a research with using a real enterprise data of some of the PAIS Lab's business partners (for example, the Exactpro Systems and London Stock Exchange)
- The work is planned to be presented on a dedicated conference
- 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

REFERENCES (1/3)

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

[2] J. Kurt. "A Brief Introduction to Coloured Petri Nets," in *Third Int. Workshop on TACAS*, Enschede, the Netherlands, Apr. 2 – 4, 1997, pp. 203 – 208.

[3] J. Kurt and M. K. Lars. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Berlin, Heidelberg, Germany: Springer, 2009.

[4] 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 Dec. 1, 2019).

REFERENCES (2/3)

[5] "Renew – The Reference Net Workshop." Renew.de. http://www.renew.de/ (accessed Dec. 1, 2019)

[6] "File:Forthandback.svg." Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Forthandback.svg (accessed Dec. 5, 2019).

[7] 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.

REFERENCES (3/3)

[8] 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 Jan. 25, 2020).

[9] "CPN Tools – A tool for editing, simulating, and analyzing Colored Petri nets." CPNTools.org. http://cpntools.org/ (accessed Dec. 1, 2019).

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

Thank you for your attention!

amrigin@edu.hse.ru
anton19979@yandex.ru

HSE Address: 20 Myasnitskaya Ulitsa, Moscow 101000 Russia | 11 Pokrovsky Boulevard, Moscow 109028 Russia