Issues in IFSs and GNs, Vol. 12, 2015/2016, 93–113

Generalized Net API

Nora Angelova[1] **and Dimitar Dimitrov**[2]

[1] Dept. of Bioinformatics and Mathematical Modelling
Institute of Biophysics and Biomedical Engineering
Bulgarian Academy of Sciences
105 Acad. G. Bonchev Str., 1113 Sofia, Bulgaria
e-mail: `metida.su@gmail.com`

[2] Faculty of Mathematics and Informatics, Sofia University
5 J. Boucher Str., Sofia-1126, Bulgaria
e-mail: `mitex@gbg.bg`

Abstract: Generalized Net Application Program Interface (GN API) is defined. The Generalized nets can be used as part of bigger algorithms and products for optimization of the parallel processes, data security etc. Therefore, the GN models must be able to create and update only by code (JAVA) and the results of each step of the algorithm execution should be available independently of the Generalized Net Integrated Development Environment (GN IDE). GN API implements these features. The paper will summarize the Generalized Nets theory, GN IDE functionality and will describe GN API functions, ideas and realization. Finally, the paper will show an example of use of the GN API.
Keywords and prhrases: Generalized nets, GN IDE, Java, GN API.
2000 Mathematics Subject Classification: 68Q85.

1 Generalized Nets

Generalized Nets (GNs) are extensions of Petri Nets [1,2]. The concept of GN was introduced in year of 1982. They are defined in a way that is principally different from the ways of defining the other types of Petri nets. In this section we give the formal definition of a Generalized Nets with priorities depend on the time (GNPDT). The main part of Generalized Net is called *transition*.

Formally, every transition is described by a seven-tuple:

$$Z = \langle L', L'', t_1, t_2, r, M, \square \rangle,$$

where:

(a) L' and L'' are finite, non-empty sets of places (the transition's input and output places, respectively) **(b)** t_1 is the current time-moment of the transition's firing;

(c) t_2 is the current value of the duration of its active state;

(d) r is the transition's *condition* determining which tokens will transfer from the transition's inputs to its outputs. Parameter r has the form of an IM:

$$r = \begin{array}{c|c} & l''_1 \; \ldots \; l''_j \; \ldots \; l''_n \\ \hline l'_1 & \\ \vdots & r_{i,j} \\ l'_i & (r_{i,j} \; - \; \text{predicate}) \\ \vdots & (1 \leq i \leq m, 1 \leq j \leq n) \\ l'_m & \end{array} ;$$

where $r_{i,j}$ is the predicate which expresses the condition for transfer from the i-th input place to the j-th output place. When $r_{i,j}$ has truth-value "$true$", then a token from the i-th input place can be transferred to the j-th output place; otherwise, this is impossible;

(e) M is an IM of the capacities of transition's arcs:

$$M = \begin{array}{c|c} & l''_1 \; \ldots \; l''_j \; \ldots \; l''_n \\ \hline l'_1 & \\ \vdots & m_{i,j} \\ l'_i & (m_{i,j} \geq 0 - - \text{ natural number or } \infty) \\ \vdots & (1 \leq i \leq m, 1 \leq j \leq n) \\ l'_m & \end{array} ;$$

(f) $\square$ is called transition type and it is an object having a form similar to a Boolean expression. It may contain as variables the symbols that serve as labels for transition's input places, and it is an expression constructed of variables and the Boolean connectives $\wedge$ and $\vee$ determining the following conditions:

$\wedge(l_{i_1}, l_{i_2}, \ldots, l_{i_u})$	$--$	every place $l_{i_1}, l_{i_2}, \ldots, l_{i_u}$ must contain at least one token,
$\vee(l_{i_1}, l_{i_2}, \ldots, l_{i_u})$	$--$	there must be at least one token in the set of places $l_{i_1}, l_{i_2}, \ldots, l_{i_u},$ where $\{l_{i_1}, l_{i_2}, \ldots, l_{i_u}\} \subset L'$.

When the value of a type (calculated as a Boolean expression) is "$true$", the transition can become active, otherwise it cannot.

The Generalized Net is called the ordered four-tuple

$$E = \langle\langle A, \pi_A, \pi_L, c, f, \theta_1, \theta_2\rangle, \langle K, \pi_K, \theta_K\rangle, \langle T, t^0, t^*\rangle, \langle X, \Phi, b\rangle\rangle,$$

where

(a) A is a set of transitions (see above);

(b) π_A is a function giving the priorities of the transitions, i.e., $\pi_A : A \to \mathcal{N}$;

(c) π_L is a function giving the priorities of the places, i.e., $\pi_L : L \to \mathcal{N}$, where

$$L = pr_1 A \cup pr_2 A$$

and obviously, L is the set of all GN-places;

(d) c is a function giving the capacities of the places, i.e., $c : L \to \mathcal{N}$;

(e) f is a function that calculates the truth values of the predicates of the transition's conditions;

(f) θ_1 is a function giving the next time-moment, for which a given transition Z can be activated, i.e., $\theta_1(t) = t'$, where $pr_3 Z = t, t' \in [T, T + t^*]$ and $t \leq t'$; the value of this function is calculated at the moment when the transition terminates its functioning. Here and below $pr_i X$ is the i-th projection of the n-dimensional set X.

(g) θ_2 is a function giving the duration of the active state of a given transition Z, i.e., $\theta_2(t) = t'$, where $pr_4 Z = t \in [T, T + t^*]$ and $t' \geq 0$; the value of this function is calculated at the moment when the transition starts functioning;

(h) K is the set of the GN's tokens. In some cases, it is convenient to consider this set in the form

$$K = \bigcup_{l \in Q^I} K_l,$$

where K_l is the set of tokens which enter the net from place l, and Q^I is the set of all input places of the net;

(i) $\pi_K : K \to N$.

(j) θ_K is a function giving the time-moment when a given token can enter the net, i.e., $\theta_K(\alpha) = t$, where $\alpha \in K$ and $t \in [T, T + t^*]$;

(k) T is the time-moment when the GN starts functioning; this moment is determined with respect to a fixed (global) time-scale;

(l) t^0 is an elementary time-step, related to the fixed (global) time-scale;

(m) t^* is the duration of the GN functioning;

(n) X is a function which assigns initial characteristics to every token when it enters input place of the net;

(o) Φ is a characteristic function that assigns new characteristics to every token when it makes a transfer from an input to an output place of a given transition;

(p) b is a function giving the maximum number of characteristics witch a given token can receive, i.e., $b : K \to N$.

2 GN IDE

Environment (GN IDE) is a software tool, which integrated GN simulation server – GNTicker. GN IDE assists the user through the whole process of modelling and simulation with GNs. The software tool allows users to load and save GN XML files, to create and edit GN models, to visualize them and to run, pause and resume simulation. GN IDE was originally written by Dimitar Dimitrov and has been developed further by Nora Angelova [3–6]. A GN model includes description of it's transitions, places, arcs, tokens, matrices (predicates), characteristic functions and data.

The Software can run on any platform with Java 71 Runtime Environment (JRE) installed. GN IDE is written in Java, hence, it is platform independent. It connects to a simulation core via the Growl Network Transport Protocol (GNTP) protocol. The GNTP, is a protocol to allow two-way communication between applications and centralized notification systems and to allow two-way communication between two machines running centralized notification systems for notification forwarding purposes.

The newest version of GN IDE implements support for JavaScript as the language of characteristic and predicates functions. For this purpose is developed EmbededSimulation class. It implements the algorithm for transition functioning when merging of tokens is permitted which support JavaScript predicates.

3 GN API Builder and GN API

The Java API requires execution of Java predicates. For this purpose the EmbededSImulation class is extended and the algorithm supports this feature. The concept of the GN API is to implement the ability to create and modify GN models, to create and control the simulation and to receive results back in convenient form only with Java code.

All of these features are available in two packages – GN API Builder and GN API.

GN API Builder contains one class GeneralizedNetBuilder.

GeneralizedNetBuilder is a class that implements all functions for creating and updating a GN model. As we mentioned above tha basic steps of generalized net implementation is adding transitions, places, tokens and their properties.

GeneralizedNetBuilder implements TransitionBuilder, PlaceBuilder and TokenBuilder classes which have an object property of the type (Transition, Place, Token) and methods for add and set their properties. GeneralizedNetBuilder includes the following public features:

- Create a GN – GeneralizedNetBuilder(String name). GeneralizedNetBuilder is constructor with a parameter of String type. It creates an object of type GeneralizedNet by name.
  - set duration of the GN functioning – GeneralizedNetBuilder setGnTime(int time). setGnTime is a setter method with a parameter of int type. It sets gloabal GN time.
  - set start simulation time – GeneralizedNetBuilder setGnTimeStart (int time). setGnTimeStart is a setter method with a parameter of int type. It sets time moment (related to the gloabal time scale) when the GN start functioning.
  - set elementary time-step – GeneralizedNetBuilder setGnTimeStep (int timeStep). setGnTimeStep is a setter method with a parameter of int type. It sets an elementary time-step, related to the gloabal time scale.
  - set token splitting property – GeneralizedNetBuilder setTokenSplittingEnabled(boolean enabled). setTokenSplittingEnabled is a setter method with a parameter of boolean type which enables or disables token splitting during the simulation. Each setter method

returns an GeneralizedNetBuilder object, allowing the calls to be chained together in a single statement without requiring variables to store the intermediate results.

  - add transtion method – TransitionBuilder addTransition(String id). addTransition is method with parameter of String type. It creates a transiton by an id, add it to GN model, creates and returns an object of type TransitionBuilder. TransitionBuilder is a class part of GeneralizedNetBuilder that implements all methods of transition setup.

- Create transitionBuilder – TransitionBuilder(Transition transition). TransitionBuilder is contructor with parameter of Transition type. It creates an TransitionBuilder object by an transition object.

  - set transition start time – TransitionBuilder setTransitionStartTime (int startTime). setTransitionStartTime is a setter method with a parameter of int type. It sets the current time-moment of the transition's firing.
  - set transition life time – TransitionBuilder setLifeTime(IntegerInf lifeTime). setLifeTime is a setter method with a parameter of IntegerInf type. IntegerInf is class that represents a natural number and allows the value "positive infinity". The method sets current value of the duration of transition active state.
  - set transition priority – TransitionBuilder setTransitionPriority(int priority). setTransitionPriority is a setter method with a parameter of int type. It sets current transition priority during the simulation.
  - set capacity – TransitionBuilder setCapacity(String fromId, String toId, IntegerInf value). setCapacity is a setter method with 3 parameters – two of String type (id of an input place and id of an output place) and one of IntegerInf type(capacity value). Method sets the capacity of transition arc from an input place to an output place. The capacity value can be "positive infinity".
  - set transition predicate – TransitionBuilder setPredicate(String fromId, String toId, JavaFunction predicate). setPredicate is a setter method with 3 parameters – two of String type(id of an input place and id of an output place) and on of JavaFunction type (predicate).

Method sets the predicate which expresses the condition for transfer from the inplut place with id fromId to the one output place with id toId.

- set transition type – TransitionBuilder setType(String type). setType is a setter method with a parameter of String type. It sets transition's type described above.

Each TransitionBuilder setter method returns an TransitionBuilder object, allowing the calls to be chained together in a single statement without requiring variables to store the intermediate results.

- add transition input place – PlaceBuilder addInput(String id). addInput is method with a parameter of String type. It creates a place by an id, add it to transition input places list, creates and returns an object of type PlaceBuilder. PlaceBuilder is a class, part of GeneralziedNetBuilder that implements all methods of place setup.
- add transition output place – PlaceBuilder addOutput(String id). addOutput is method with a parameter of String type. It creates a place, add it to transition output places list, creates and returns an object of type PlaceBuilder.

• Create placeBuilder object – PlaceBuilder(Place place, Transition lastTransition). PlaceBuilder is constructor with 2 parameters – one of Place type and one of Transition type. It creates an PlaceBuilder object for an place and place transition.

  - set place capacity – PlaceBuilder setPlaceCapacity(IntegerInf capacity). setPlaceCapacity is a setter method with a parameter of IntegerInf type. It sets the place capacity. The capacity value can be – positive infinity".
  - set place priority – PlaceBuilder setPlacePriority(int priority). setPlacePriority is a setter method with a parameter of int type. It sets current place priority during the simulation.
  - set place merge rule – PlaceBuilder setMergeRule(JavaFunction function). setMergeRule is a setter method with a parameter of JavaFunction type. It sets a mergeRule which enables or disables tokens merge in the place during the simulation.

- set place merge boolean value – PlaceBuilder setMergeTokens (boolean merge). setMergeTokens is a setter method with a parameter of boolean type which enables or disables tokens merge in the place during the simulation.
- set place charachteristic function – PlaceBuilder setCharFunction (JavaFunction charFunction). setCharFunction is a setter method with a parameter of JavaFunction type. It sets a characteristic function that assigns new characteristics to each token when it enters in the place.

Each PlaceBuilder setter method returns an PlaceBuilder object, allowing the calls to be chained together in a single statement without requiring variables to store the intermediate results.

- add token – TokenBuilder addToken(String id). addToken is a method with a parameter of String type. It creates a token by an id and current place, add it to the model, creates and returns an object of type TokenBuilder. TokenBuilder is a class, part of GeneralizedNetBuilder that implements all methods of token setup.
- add periodic token generator – TokenBuilder addPeriodicTokenGenerator(String id, int period). addPeriodicTokenGenerator is a method with two parameters of String and int type. It creates a token generator by an id and time period that generates a token for this place for each period.
- add random token generator – TokenBuilder addRandomTokenGenerator(String id). addRandomTokenGenerator is a method with a parameter of String type. It create token generator that creates and token for this place randomly.
- add conditional token generator – TokenBuilder addConditionalTokenGenerator(String id, JavaFunction condition). addConditionalTokenGenerator is a method with two parameters of String and JavaFunction type. It creates token generator that generate token when condition is true.

• Create tokenBuilder object – TokenBuilder(Token token, Place lastPlace, Transition lastTransition). Token Builder is constructor with 3 parameters – one of Token type, one of Place type and one of Transition type.

It creates an TokenBuilder object for an token, token place and token transition.

- set token priority – TokenBuilder setTokenPriority(int priority). setTokenPriority is a setter method with a parameter of int type. It sets current token priority during the simulation.
- set token entering time – TokenBuilder setTokenEnteringTime(int enteringTime). setTokenEnteringTime is a setter method with a parameter of Int type. It sets a time-moment when a given token can enter into the GN model.
- set token leaving time – TokenBuilder setTokenLeavingTime (IntegerInf leavingTime). setTokenLeavingTime is a setter method with a parameter of IntegerInf type. It sets a time-moment when a given token can leave the GN model.
- add token characteristic – TokenBuilder addCharacteristic(String name, String type, int history). addCharacteristic is method with 3 parameters – two of String type and one of int type. The method add token characteristic with name, type and history.
- add token characteristic with value – TokenBuilder addCharacteristic(String name, String type, int history, String value). addCharacteristic is method with 4 parameters – three of String type and one of int type. The method add token characteristic with name, type and history and set its value with string.

Each TokenBuilder method returns an TokenBuilder object, allowing the calls to be chained together in a single statement without requiring variables to store the intermediate results.

Once the GN is completed, it should be build and prepare to start. GeneralizedNetBuilder include JavaGeneralizedNet build() method witch finally creates an JavaGeneralizedNet object from an GeneralizedNetBuilder.

The GN API implements the ability to create and control the simulation and to receive results back in convenient form. GN API includes the following classes:

- GeneralizedNetFacade – GeneralizedNetFacade is a class that implements start simulation functionality. It works only with GN objects of JavaGeneralizedNet type created from the GeneralizedNetBuilder. GeneralizedNetFacade includes 2 methods for start simulation.

  - start simulation – JavaSimulation startSimulation(JavaGeneralizedNet gn). startSimulation method accepts one parameter of JavaGeneralizedNet type and returns an object of JavaSimulation type. The method creates a simulation events listener and calls startSimulation method with two parameters.
  - start simulation with events listener – JavaSimulation startSimulation(JavaGeneralizedNet gn, SimulationEventsListener listener). The method has 2 parameters – one of type JavaGeneralizedNet type and one of type SimulationEventsListeners. It creates a simulation object, add observer, start simulation and return simulation object.

- JavaSimulation – JavaSimulation is a class that extends EmbededSimulation class [3]. JavaSimulation class implements the ability to create and control the simulation. It has one constuctor – JavaSimulation(GeneralizedNet gn) with a parameter of GeneralizedNet type. The simulation is carried out in steps. Steps are initiated by void step(int count) method. It has a parameter of type int that says how many steps of the simulation to be released.

- SimulationEventsListener – SimulationEventsListener is a class that implements simaltion events observer.

  - create SimulationEventsListener – SimulationEventsListener(). SimulationEventsListener is default constructor that creates an observer with an update method. The update method handles 3 type of events – JavaEnterEvent, JavaMoveEvent and JavaLeaveEvent.
  - get events observer – BaseObserver getObserver(). getObserver is a method with no parameters that returns the events observer.

- JavaGnEvent – JavaGnEvent is a class that implements base event functionality. It has an protected GnEvent object and two methods.

- get event token – JavaToken getToken(). getToken is a getter method with no parameters which returns token associated with the event.
- get event characteristics – List <JavaCharacteristic >getChars(). getChars is a getter method with no parameters which returns a list with javaCharacteristics associated with the event.

• JavaEnterEvent – JavaEnterEvent is a class that implements place entering event. The event is fire when a token enters the GN in a place. JavaEnterEvent class extends JavaGNEvent class described above retains all methods of it and add a constructor and getPlace method.

  - create JavaEnterEvent – JavaEnterEvent(EnterEvent event).JavaEnterEvent is a constructor with one parameter of EnterEvent type. It creates an object of JavaEnterEvent type.
  - get event place – JavaPlace getPlace(). getPlace is a getter method with no parameters which returns a place where the token is entering during the simulation step.

• JavaLeaveEvent – JavaLeaveEvent is a class that implements place leaving event. The event is fire when a token leave the GN from a place. JavaLeaveEvent class extends JavaGNEvent class described above retains all methods of it and add a constructor and getPlace method.

  - create JavaLeaveEvent – JavaLeaveEvent(LeaveEvent event).JavaLeaveEvent is a constructor with one parameter of LeaveEvent type. It creates an object of JavaLeaveEvent type.
  - get event place – JavaPlace getPlace(). getPlace is a getter method with no parameters which returns a place where the token is leaving the GN.

• JavaMoveEvent – JavaMoveEvent is a class that implements place moving event. The event is fire when a token move from one place to other. JavaMoveEvent class extends JavaGNEvent class described above retains all methods of it and add a constructor, getStartPlace and getEndPlace methods.

  - create JavaMoveEvent – JavaMoveEvent(MoveEvent event). JavaMoveEvent is a constructor with one parameter of MoveEvent type. It creates an object of JavaMoveEvent type.

  - get start event place – JavaPlace getStartPlace(). getStartPlace is a getter method with no parameters which returns a place which the token leaves during movement.
  - get end event place – JavaPlace getEndPlace(). getEndPlace is a getter method with no parameters which returns a place where the token enters during movement.

GN API implements classes that allows to create Java objects from engine objects and to get their properties. The API differs the following objects – GN, transition, place, token, characteristic or function. We are described all classes below.

- JavaGeneralizedNet – JavaGeneralizedNet is a class with one private property of GeneralizedNet type. It implements constructor and getter methods associated with the Java(API) GN model.
  - create JavaGeneralizedNet object – JavaGeneralizedNet(GeneralizedNet gn). JavaGeneralizedNet is a constructor with one parameter of GeneralizedNet type. It creates an object of JavaGeneralizedNet type.
  - get name – String getName()
  - get current time – int getTime()
  - get all transitions – List<JavaTransition>getTransitions()
  - get transition by id – JavaTransition getTransition(String id)
  - get all places – List<JavaPlace>getPlaces()
  - get place by id – JavaPlace getPlace(String id)
  - get tokens – List<JavaToken>getTokens()
  - get token by id – JavaToken getToken(String id)
- JavaTransition – JavaTransition is a class with one private property of Transition type. It implements constructor and getter methods associated with the Java(API) Transition.
  - create JavaTransition object – JavaTransition(Transition transition). JavaTransition is a constructor with one parameter of Transition type. It creates an object of JavaTransition type.

- check transition equality – boolean equals(Object obj)
- get id – String getId()
- get priority – int getPriority()
- get input places – List<JavaPlace>getInputs()
- get output places – List<JavaPlace>getOutputs()

• JavaPlace – JavaPlace is a class with one private property of Place type. It implements constructor and getter methods associated with the Java(API) Place.

  - create JavaPlace object – JavaPlace(Place place)
  - check place equality – boolean equals(Object obj)
  - get id – String getId()
  - get priority – int getPriority()
  - get capacity – IntegerInf getCapacity()
  - get transition where the place is input – JavaTransition getInput()
  - get transition where the place is output – JavaTransition getOutput()
  - get tokens – List<JavaToken>getTokens()
  - get token by id – JavaToken getToken(String id)
  - add token with id – JavaToken addToken(String id)
  - remove token by id – void removeToken(String id)

• JavaToken – JavaToken is a class with one private property of Token type. It implements constructor and getter methods associated with the Java(API) Token.

  - create JavaToken object – JavaToken(Token token)
  - check token equality – boolean equals(Object obj)
  - get id – String getId()
  - get priority – int getPriority()
  - get host place – JavaPlace getHost()
  - get token characteristics – List<JavaCharacteristic>getChars()

- get default characteristics – JavaCharacteristic getDefault()
- get characteristic by id – JavaCharacteristic getChar(String id)
- add characteristic – JavaCharacteristic addChar(String id, String type, int history)
- delete characteristic by id – void delChar(String id)

• JavaCharacteristic – JavaCharacteristic is a class with one private property of Characteristci type. It implements constructor, getter and setter methods associated with the Java(API) Characteristic.

- create JavaCharacteristic object – JavaCharacteristic(Characteristic characteristic)
- get name – String getName()
- get type – String getType()
- get value – String getValue()
- set value – void setValue(String value)
- push historic value – void pushValue(String value)
- get history – List<String>getHistory()

• JavaFunction – JavaFunction is a class that implements constructor and run method for Java function objects. Java Functions are important for the predicates definitions.

- create JavaFunction by name – JavaFunction(String name)
- run method – abstract Object run(GeneralizedNet gn, JavaToken token)

• JavaFunctionReference – JavaFunctionReference is a class that implements reference to the function. It has one private property – JavaFunction, constructor and a get method.

- create reference function object – JavaFunctionReference(JavaFunction function)
- get function – JavaFunction getFunction()

• JavaFunctionRunner – JavaFunctionRunner is a class that implements function runner functionality. It has one private and static instance property, get and run method.

- get instance – static JavaFunctionRunner getInstance(). The method try to get instance, if it is not defined, the method creates the new one and returns it.
- run JavaFunctionReference – Object run(FunctionReference function, GeneralizedNet gn, Token token). It has 3 parameters of FunctionReference, GeneralizeNet and Token type. The first parameter is function. The method cast FunctionReference to JavaFunctionReference type, get function and run it with gn and JavaToken parameter.

4 Wastewater Treatment Process Simulation Using GN API

There have been developed several generalized net models of some wastewater treatment systems (see [8, 9]. The first GN model developed using GN IDE software for the purposes of wastewater treatment process simulation based on real experiment data is presented in [10]. The GN model describes technological scheme of wastewater treatment including the three main stages: mechanical, physics-chemical and biological treatment. The detailed description, from the point of view of wastewater treatment, is given in [10].

The flow of treated water gets into the exit position as purified water and theated waste. The model has 7 transtions and 19 places. Six of places – l_3, l_6, l_9, l_{12}, l_{14}, l_{17} are using for monitoring and storing information of the process. The rest described the process of water purification from different elements like oil, mechanical impurities ets. In this chapter, the part of GN model of wastewater treatment plant will be implemented with help of GN API.

The graphical representation of WT process which will be implemented is shown in Fig. 4.

The model has 7 transitions. Each of them described step of water treatment. First it should create a generalized net. For this purpose it is necessary to use GeneralizedNetBuilder constructor.

Example:

```
GeneralizedNetBuilder wastewaterTreatmentGN = new
   GeneralizedNetBuilder("WTGeneralizedNet")
```

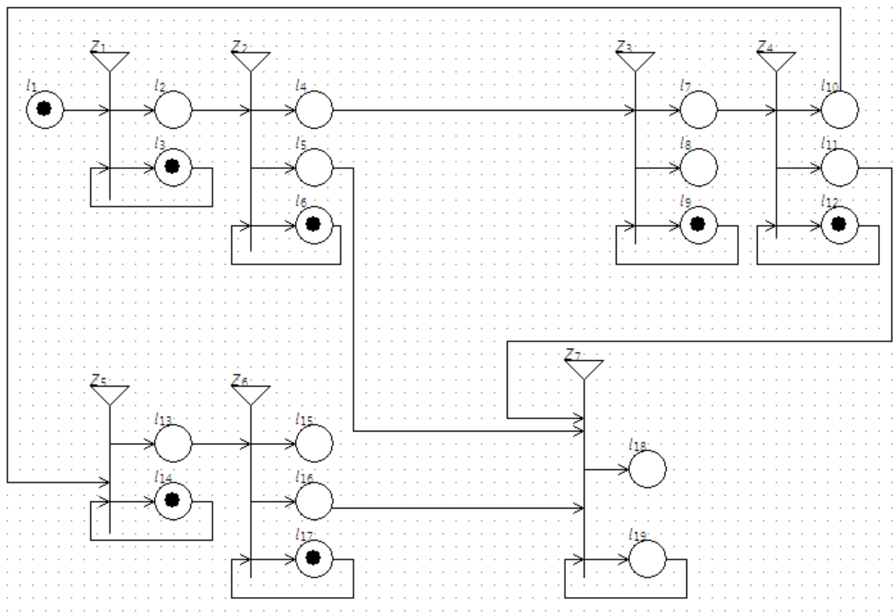

Figure 1: Graphical representation of the GN model of WT process

First Transition is named Z_1. It has two input places – l_1, l_3 and two ouput places – l_2, l_3. The place l_1 contains conditional token generator that creates a new token for each record of previously collected data. Each token has a characterstic wastewater quantity. l_3 contains token which store information like a wastewater quantity history characteristic during the simulation.

The r_1 matrix allows token movement from l_1 to l_2 and from l_3 to l_3

First it should create and add transition with name Z_1.

Example:

```
TransitionBuilder transitionZ1 =
   wastewaterTreatmentGN.addTransition("Z1");
```

Then it should add input place l_1.

Example:

```
PlaceBuilder placeL1 = transitionZ1.addInput("l1");
```

Then it should add conditional token generator.

Example:

```
TokenBuilder generatedToken =
    placeL1.addConditionalTokenGenerator("tokenGeneratorL1",
    new JavaFunction("isGenerate") {
   @Override
   public Object run(GeneralizedNet net, JavaToken
       token) {
   // some condition (days number if the simulation
       works with days data during a month or etc.)
   }
})
```

Each generated token can be changed like set entering, leaving time, priority or start characterstic.

Then it should add other input place l_3 with token "l3_waterQuantity_token" that will record information about the polutted water quantity during the simulation. The token will not leave the Gn and the leaving time will be infity (default state). The token has a characteristic with name "Q3_history" of type "number" and history. In contrast to examples above, the definition will be on a single line.

Example:

```
transitionZ1.addInput("l3").addToken("l3\_waterQuantity\
_token").addCharacteristic("Q3\_history", ``number",
    history\_capacity);
```

Now, it should add two ouput places l_2 and l_3, two characteristic functions for l_2 and l_3 and the transition predicat that determines under what conditions the token can pass from input to ouput places. A token will be enter in place l_2, if water is polluted and will obtain a new characteristic state with a value "Polluted wastewater for treatment". The token in position l_3 – "l3_waterQuantity_token" will pass from l_3 to l_3 if has polluted water and will obtain a new value for "Q3_history" characteristic. The definion will be on a single line.

Example:

```
transitionZ1.addOutput("l2").setCharFunction(new
    JavaFunction("l2\_charFunc") {
   @Override
   public Object run(GeneralizedNet net, JavaToken
       token) {
     token.addChar("state", ``String",
        100).setValue("Polluted wastewater for
        treatment");
   }
}).addOutput("l3").setCharFunction(new
    JavaFunction("l3\_charFunc") {
   @Override
   public Object run(GeneralizedNet net, JavaToken
       token) {
       token.getChar("Q3\_history").setValue('some value
          from our data');
       return true;
   }
}).setPredicate("l1", ``l2", new JavaFunction("l12") {
  @Override
  public Object run(GeneralizedNet net, JavaToken
      token) {
    // return true if water is polluted and false
       otherwise
  }
}).setPredicate("l3", ``l2", new JavaFunction("l32") {
  @Override
  public Object run(GeneralizedNet net, JavaToken
      token) {
    return false;
  }
}).setPredicate("l1", ``l3", new JavaFunction("l13") {
  @Override
  public Object run(GeneralizedNet net, JavaToken
      token) {
    return false;
  }
}).setPredicate("l3", ``l3", new JavaFunction("l33") {
  @Override
```

```
  public Object run(GeneralizedNet net, JavaToken
      token) {
    // return true if has water in $l_{1}$ and water is
        polluted and false otherwise
  }
});
```

Similarly can be described each transition in the GN. The GN API allows to user to make model definition on single line. When the model is described it should be build and convert to JavaGeneralizedNet type. To start the simulation is necessary to use either of the metods in GeneralizedNetFacade class. Both metods return JavaSimulation object and has a parameter of type JavaGeneralizedNet. The second metod has an event listener parameter. Below we will show the use of both metod.

Example1:

```
JavaSimulation simulation =
    GeneralizedNetFacade.startSimulation(wastewaterTreat-
    mentGN);
```

Example2:

```
JavaSimulation simulation =
    GeneralizedNetFacade.startSimulation(wastewaterTreat-
    mentGN, new SimulationEventsListener() {
  @Override
  public void handleEvent(List<JavaGnEvent> events) {
    for (JavaGnEvent event: events) {
      System.out.println("event: `` +
          event.getClass().getSimpleName() + `` `` +
          event.getToken().getId() + `` `` +
          event.getChars().size());
    }
  }
});
```

When the simulation is sterted we can play a step or more. Example: simulation.step(1);

If it use the second metod to start the simulation (with events listener), for each step of simulation progress, it will call the handleEvent function with all events for the step.

This allows programmers to process and store this information in the most convenient way. Furthemore the most part of the models can be simplified.

Let us look at the example again.

As we mentioned above, the model described wastewater treatment process. The most important part of the net is detecting to unusual measurements, to view historic data for wasteWater quantity (place l_2) etc. Now, all data is available in the events handlers, we can parse the information for tokens which are entering in place l_2 for example, to get the water quantity and to store it to database, arrays etc. The great advantage is that this information can be used like a history, can be visualized from different interfaces, to be analyzed from diffenrent algorithms without the limitations of the Genedit and the graphics editor. As a consequence the model can be simplified by removing l_3, l_6, l_9, l_{12}, l_{14}, l_{17} places and tokens with host in this places.

5 Benefits and Conslusion

The difference between models is that the first one simulates WT process during different months using previously collected data, has graphical representation and not communicate with other application. The second one is more appropriate for the general case. The main benefits of GN API implementation are:

- a simple Java implementation
- use GN models functionality only through the interface
- learning of the Genedit is not necessary
- ability to be used as part of bigger and real-time application
- sending of events occurring in the GN model through JAVA code that can easily be processed to detect unusual measurements
- simplified models
- opportunity for a different visualization of stored data.

References

[1] Murata, T.: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*. Vol. 77, 1989, No. 4, 541–580.

[2] Petri, C.-A. *Kommunication mit Automaten,* Ph. D. Thesis, Univ. of Bonn, 1962; Schriften des Inst. fur Instrument. Math., No 2, Bonn, 1962.

[3] Dimitrov, D. G. GN IDE – A Software Tool for Simulation with Generalized Nets. *Proceedings of Tenth Int. Workshop on Generalized Nets*. Sofia, 5 December 2009, 70–75.

[4] Dimitrov, D. G. A Graphical Environment for Modeling and Simulation with Generalized Nets. *Annual of "Informatics" Section*, Union of Scientists in Bulgaria, Vol. 3, 2010, 51–66 (In Bulgarian).

[5] Dimitrov, D. G. Software products implementing generalized nets. *Annual of Section "Informatics"*, Union of Scientists in Bulgaria, Vol. 3, 2010, 37–50 (In Bulgarian).

[6] Andonov, V., N. Angelova. Modifications of the algorithms for transition functioning in GNs, GNCP, IFGNCP1 and IFGNCP3 when merging of tokens is permitted. Imprecision and Uncertainty in Information Representation and Processing, Vol. 332 Studies in Fuzziness and Soft Computing, Springer, 2016, 275–288.

[7] Trifonov, T., K. Georgiev. GNTicker – A software tool for efficient interpretation of generalized net models, *Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, Vol. 3, Warsaw, 2005, 71–78.

[8] Georgieva V., E. Sotirova. Generalized Net Model of Biological Treatment of Wastewater, *Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, Vol. 11, 2014, 63–72.

[9] Georgieva V., O. Roeva, T. Pencheva. Generalized Net model of Physics-chemical Wastewater Treatment, *Ecology & Safety*, Vol. 9, 2015, 468–475.

[10] Georgieva V., N. Angelova, O. Roeva, T. Pencheva. Simulation of Parallel Processes in Wastewater Treatment Plant Using Generalized Net Integrated Development Environment, *Comptes rendus de l'Academie bulgare des Sciences*, Tome 69, 2016, No. 11, 1493–1502.