ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2015, T. 44, Nr. 1

Topology Based Automatic Formal Model Generation for Point Automation Systems

Muhammet Ali Nur Oz, Ibrahim Sener, Ozgür Turay Kaymakcı, Ilker Ustoğlu, Galip Cansever

Department of Control and Automation Engineering, Yıldız Technical University, Istanbul, 34220, TURKEY
e-mail: maoz@yildiz.edu.tr, isener@yildiz.edu.tr, kaymakci@yildiz.edu.tr, ustoglu@yildiz.edu.tr, cansever@yildiz.edu.tr

crossref http://dx.doi.org/10.5755/j01.itc.44.1.7382

Abstract. Designing and developing a point automation system is a challenging task since railway transportation systems are required to be highly secure and safe systems. Nowadays point automation systems are usually designed manually, this results in a waste of personnel, time and resources. So in this study, we developed and established a software tool in order to automatically generate formal models for point automation systems. The novelty of our study is that our models are created automatically by a software. Here designing time and human errors are reduced to a minimum thus safe, reliable and secure system models are generated. The developed software has a built in graphical interface which is used to model the basic station topology and using this model, software generates a point automation system's Timed-Arc Petri Net (TAPN) models, which is a strongly recommended formal method by CENELEC EN50128 standard, automatically. Generated TAPN models are also verified automatically for specified safety requirements by using Computational Tree Logic (CTL), which is also a formal proof method strongly recommended by CENELEC EN50128 standard. The TAPN models were automatically generated and verified with 100% success by taking the point automation systems of stations on M1 Aksaray-Airport line, operated by Istanbul Transportation Co., as the reference.

Keywords: Point automation; Timed-arc Petri net; Automatic model generation; Formal verification; Interlocking; Railway systems.

1. Introduction

Transportation has become one of the most important concerns for people living at cities nowadays. People can travel from one place to another more economical, safer and faster through the railway systems in urban and interurban transportation. Considering that railway transportation has such advantages, it is seen that the railway systems have a great superiority to other modes of transportation. The safe journey to be guaranteed against any collision or accidents in railway transportation system is very important. A small error at the railway system, which may occur, can cause very serious consequences such as loss of human lives, severe injuries, considerable economic penalties and environmental damages. Because of these dangers, railway safety systems like interlocking, signalization and point automation systems are essentially designed within strict rules and binding standards. These systems are usually realized based on the CENELEC (Comite Europeen de Normalisation Electrotechnique) EN5012x family of railway standards including EN50126, EN50128 and EN50129 standards, which concentrate on the modeling methods necessary for ensuring safety and reliability in railway transportation systems. These standards apply to both heavy rail systems and light rail systems [1].

Formal methods, which are based on mathematical foundation, are strongly recommended to be utilized in the modeling and verifying of signalization and interlocking systems for railway applications by CENELEC EN50128 (Table A.17). The reliability and robustness of a designed system can be increased using these methods. There exist a great number of studies in literature regarding the designing and verifying of signalization and interlocking systems using formal methods. Zafar [2] formed a formal model for railway moving interlocking system by using Z notation, which is a formal modeling language. In a study [3], Winter used CSP (Communicating Sequential Processes) and also checked the functional specifications of the formal model using FDR (Failures-Divergences Refinement) model checking tool. Banci and Fantechi [4] modeled a railway interlocking system using state charts. In our previous study [5], we formed a formal model for point automation system by using Timed-Arc Petri Nets, which is a recommended formal modeling method by the relevant standard. Moreover, identified safety requirements for the automation of the points were verified through CTL (Computational Tree Logic), which is one of the recommended formal proof methods by the concerned standard. Also for detailed information about TAPN refer to [6 - 9].

The generation of formal models of system is currently a manual process, which is inefficient and error-prone due to the complexity of the railway yard, where there are many points and routes, and human interferences. Adding to the fact that mistakes in design can lead to serious accidents resulting in loss of many lives not to mention financial losses. For this reason, the automatic generation of models, which describe the system, from the railway topology and the automatic verification of identified safety requirements using formal proof methods according to generated models based on the topology is very significant. The main advantage of automatic modeling is to significantly reduce the human errors and improve the efficiency in the generation and verification of formal models of system. Automatic generated models are generally more reliable and so the reliability and safety of the whole system increases. Automatic generation of the models has a great importance in order to minimize the modeling faults.

Moreover, there exist a limited number of studies in literature regarding the automatic generation and verification of models that describe the system as well as software tool, which can be used for railway transportation system. Interlocking tables were generated from the station topology automatically in some studies [10, 11] but here the obtained interlocking tables were not verified. Cao et al. [12] developed a tool, which can be used to automatically generate and verify the interlocking table of railway station designed by DSL-CBI (Domain Specific Language for Computer Based Interlocking). In a study [13], Sachdev et al. a software application that can be used to automatically generate interlocking schemes for substations and also tested them. In [14], a component-based model, which is used to describe the topology of the station, was introduced. In another study [15], a method for the automatic generation of application data for interlocking simulation system based on GIS infrastructure data was proposed, and the design of the complementary software tool was described. Haxthausen [16] described a tool for extracting formal safety conditions from interlocking tables for relay interlocking systems.

This study focuses on the automatic generation of the models from the station topology for automation and control of the points. In this study, using a software tool, which was developed by using C# programming language, station TAPN models were formed automatically. The development of station TAPN models from the topology is simplified thanks to this generalized tool. Automatically generated models are stored in XML format. These models can be viewed using TAPAAL [17], which is a tool for modeling, simulation and verification of Timed-Arc Petri nets. Another important issue is to test whether the models, which were formed automatically from the station topology to ensure the accurate and safe conduct of the point automation system, fulfill the identified safety requirements or not. Therefore, TAPAAL editor was used to verify the existence of anticipated safety requirements for the relevant models. The verification of the identified safety requirements was made automatically through CTL, which is also recommended formal proof methods by CENELEC EN50128.

The paper is organized as follows. In section 2, the description of formal modeling of point automation system subcomponents is given. Automatic generation of formal models for point automation system is introduced in section 3. Automatic generation of formal models of Bastabya Station is presented in section 4. Verification of automatic generated formal models for point automation system of Bastabya Station is given in section 5. Software performance for different stations is shown followed by conclusion.

2. Formal Modeling of Point Automation System Subcomponents

Complex systems can be obtained by assembling simpler components, which are building blocks of the system. The blocking blocks for point automation system are points, signals and track circuits. Points are mechanical tools, which are usually controlled with an electrical motor. They are movable components, which guide the trains towards from one line to another at a railway intersection according to the desired route. So, they play a crucial role in ensuring a safer and speedier journey. A point has generally two positions, which can be settled, named as normal and diverging. The correct position of points according to the desired route is fundamental to the safe running of a railway. In case of any wrong position, two trains may be on the same track and they may crush each other. Efficiency and speed of a railway is highly affected by the number and form of the points. Reliability and safety of a railway is also directly relevant to the automation and controlling of these points. For all these reasons, automation and controlling of points are extremely significant, even indispensable. The major purpose of conducting point automation is to considerably minimize the human errors and improve the efficiency.

The TAPN model, which was formed for points in [5], was used as point model. Point model consists of six places and four transitions. It is accepted that all points in the station are at normal position at the initial stage. For changing the position of point, it should be enabled (P_Enable), which means the point is not locked for any route and there should be no tokens in the TCM place, which means the point is not occupied by a train. When enabled, the point at normal position (Point_N) moves towards diverging position (Point_R). It is required to reach diverging position by completing its movement within a certain time period interval ([max1, max2]). In case it does not achieve diverging position within [max1, max2] time interval, this will be identified as point position error and the intended route is not opened. The same rule applies for the point at diverging position when it moves from diverging position to normal position. The relevant TAPN model formed can be seen in Fig. 1. Here Table 1 also represents the definitions of places in point model.

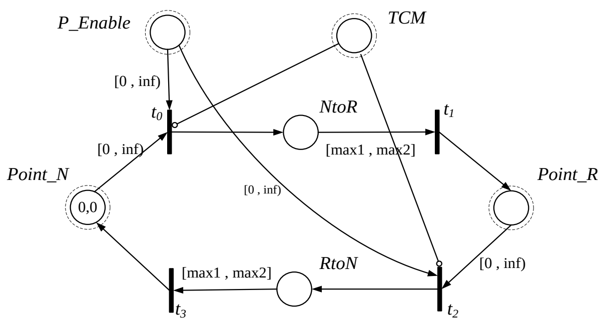

Figure 1. Point timed-arc Petri net model

Table 1. Definitions of places in point model

Place	Definition
Point_N	Point is in normal position
Point_R	Point is in diverging position
P_Enable	Point can change its position
NtoR	Point goes from normal to diverging
RtoN	Point goes from diverging to normal
TCM	Point is occupied

Other components such as signals and track circuits also play an important role in the conducting of point automation at a station. Railway signals are a system used to control railway traffic safely so that collision of the trains can be prevented. Signals transmit colored light (green, red, yellow) notice, which notifies the trains regarding the proceeding of the trains and feed up until the next signal. It becomes necessary to use signals, which enable a safe area between the trains, when the brake distance of railway transportation vehicles is taken into consideration. The TAPN model, which was formed for signals in [5], was used as signal model. Signal model consists of four places and two transitions. It is accepted that all signals are red at the initial stage. After the points on the route to be opened achieve the relevant position, the signal is enabled and green notification is transmitted to the train for allowing pass. As the train passes the signal and occupies the first track circuit (TrEntM), the signal indicates red once again. TAPN model formed for the signal can be seen in Fig. 2. Table 2 also represents the definitions of places in signal model.

It is important to know at which point the trains are so that railway traffic can be managed safely. Track circuit is a simple electrical circuit designed to detect the absence or presence of a railway vehicle in a certain part of a railway. They provide information whether the route is available or occupied by a railway vehicle. The basic principle of a track circuit is based on short circuiting the rails by the train wheels and axles. If there is a short circuit between rails in a part of the railway, it is understood that there is a train in this part and so any other trains are not allowed to enter this track. Otherwise, namely if there is no train on the track, it is understood that the track is safe to set a route and permit a train to proceed.

CENELEC EN 50128 Table A.4-Software Design & Imp. requires the use of Modular Approach in modeling and designing of railway systems. Modular approach enables us to analyze and define the system in elementary pieces as well as facilitation of the modeling. In order to model and design of the system on modular basis, separate TAPN models were formed for point and signal as shown above. No standard model was formed for the track circuit, because it changes according to desired route.

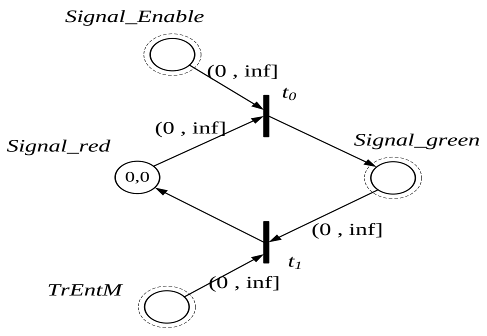

Figure 2. Signal timed-arc Petri net model

Table 2. Definitions of places in signal model

Place	Definition
Signal_red	Signal indicates red
Signal_green	Signal indicates green
Signal_Enable	Signal is enabled
TrEntM	Train enters the first track circuit

3. Automatic Generation of Formal Models for Point Automation System

A software tool was developed to generate automatically the system TAPN models from the railway station topology. C# programming language, which is recommended by CENELEC EN 50128, was used in development of the tool. The specially developed software consists of two parts, a graphical user interface, which allows users to be able to draw the station topology and application software, which generates the system TAPN models and stores them in a XML file.

Users don’t need to write an additional program through this tool in order to generate the relevant models. The main task of the user is to draw the station topology and add the necessary components to the station diagram. Flowchart shows the automatic model generation stages in Fig. 3.

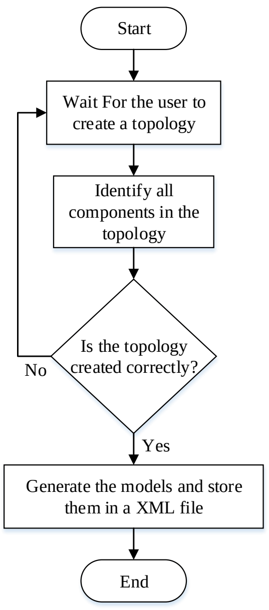
Start
Wait For the user to create a topology
Identify all components in the topology
Is the topology created correctly?
No
Yes
Generate the models and store them in a XML file
End

Figure 3. Flowchart of topology drawing

3.1. Graphical User Interface

In order to create an automatic pattern net model for stations, users need to specify a graphical representation of the station, namely its single line diagram. To solve this problem, a graphical user interface, which provides a simple way for user to draw stations topology, was developed. Graphical user interface consists of a layout editor, a toolbox and an information panel. The layout editor is used to display the represented graphical model. It is divided into grids to enable simple clicks and create actions. The toolbox, which includes point, track and signal, allows the modeler to choose the component in order to add to the graphical model. All the components, which are successfully added to the layout, are labeled such as P = {P1, P2,…Pn} for points, T={T1, T2,…Tn} for track circuits and S={S1, S2,…Sn} for signals. These labels are shown in the information panel.

In order to model the station, the user must know the number of tracks, the number of points and where these points connect to the tracks. The user must go through four steps in order to complete the graphical representation model. Tracks are created in the first step by choosing the track icon on the toolbox and later specifying the start and end locations by clicking on the layout editor. Secondly points are created. To do this, modeler chooses the point icon from the toolbox and then specifies the locations where the point intersects with tracks. The third step is to build signals. Signal is created in the same way like point by clicking the signal icon from the toolbox and specifying its location on the track. In the last step, the modeler has to point out where the train can enter the station. To do this, the modeler must first click on the specify icon and later click on the locations where the train can enter the station. Graphical modeling of the station is finished by clicking the end button. After the successful creation of the output file, a massage appears at the bottom side of the interface. A screenshot of the interface with an example station model can be seen in Fig. 4.

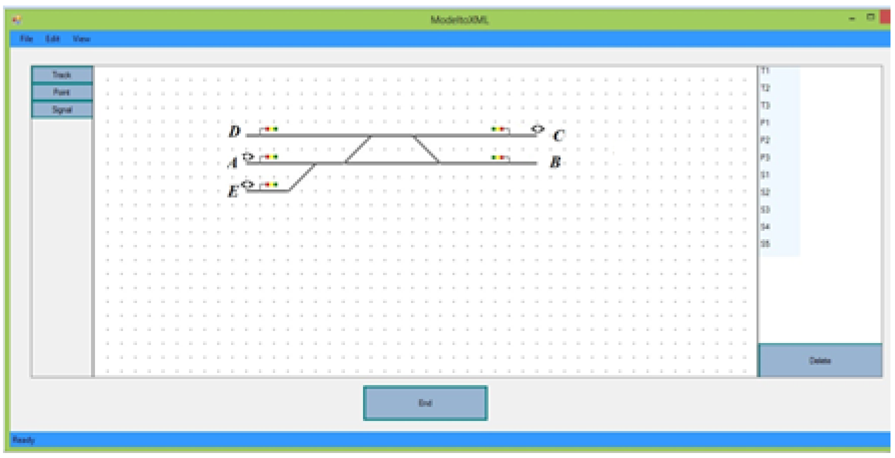

Figure 4. Screenshot of the interface

3.2. Application Software

The application software is developed by using C#, which is an object oriented programming language. The main purpose of the application software is to create the system model using information, which comes from the station topology in the process of automatic generation of the models. The validated information by graphical interface, belonging to station, is transferred to the application software. The application software generates the route and track circuit TAPN models and stores in XML file including all point and signal models. Automatic generated models can be seen by using TAPAAL, which is a tool for modeling, simulation and verification of TAPN.

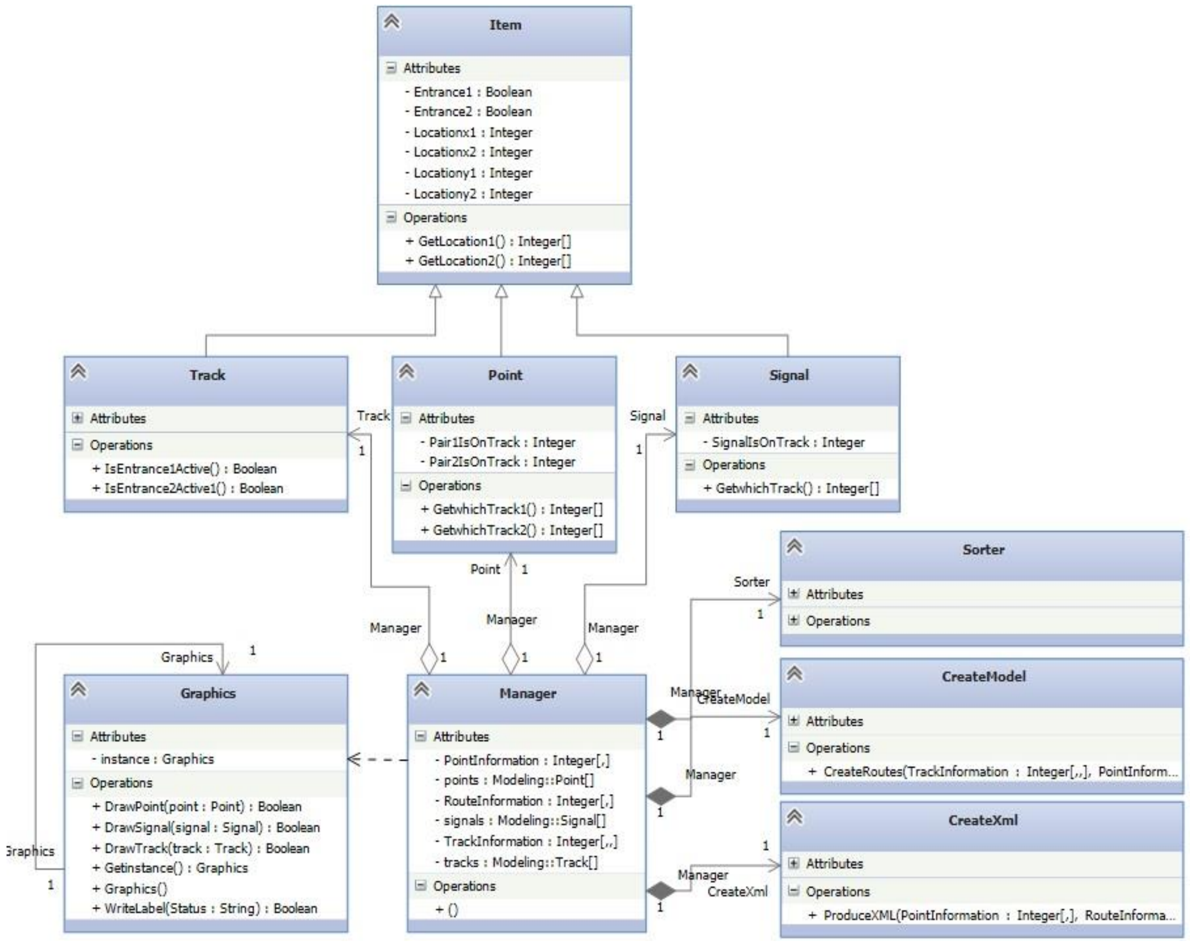
Item
Attributes
- Entrance1 : Boolean
- Entrance2 : Boolean
- Locationx1 : Integer
- Locationx2 : Integer
- Locationy1 : Integer
- Locationy2 : Integer
Operations
+ GetLocation1() : Integer[]
+ GetLocation2() : Integer[]
Track
Attributes
Operations
+ IsEntrance1Active() : Boolean
+ IsEntrance2Active1() : Boolean
Point
Attributes
- Pair1IsOnTrack : Integer
- Pair2IsOnTrack : Integer
Operations
+ GetwhichTrack1() : Integer[]
+ GetwhichTrack2() : Integer[]
Signal
Attributes
- SignalIsOnTrack : Integer
Operations
+ GetwhichTrack() : Integer[]
Sorter
Attributes
Operations
CreateModel
Attributes
Operations
+ CreateRoutes(TrackInformation : Integer[,,], PointInform...
CreateXml
Attributes
Operations
+ ProduceXML(PointInformation : Integer[,], RouteInforma...
Graphics
Attributes
- instance : Graphics
Operations
+ DrawPoint(point : Point) : Boolean
+ DrawSignal(signal : Signal) : Boolean
+ DrawTrack(track : Track) : Boolean
+ Getinstance() : Graphics
+ Graphics()
+ WriteLabel(Status : String) : Boolean
Manager
Attributes
- PointInformation : Integer[,]
- points : Modeling::Point[]
- RouteInformation : Integer[,]
- signals : Modeling::Signal[]
- TrackInformation : Integer[,,]
- tracks : Modeling::Track[]
Operations
+ ()
Track
Signal
Point
Manager
Sorter
CreateModel
CreateXml
Graphics
1

Figure 5. UML model of the program

Also for detailed information about TAPAAL refer to [13, 14]. The UML model of the application software can be seen in Fig. 5.

Application software's duties can be divided into four major sections. Managing the interface is the first duty. All of the components, which are created in the station representative model by the graphical user interface, are created inside the program as instances of component classes such as point, signal and track circuit classes. In order to create these component class instances, a manager class is used. Thus, if a new component is created or edited in the interface, corresponding instances of component class can be updated. Since graphics class should only be created once and should be accessible from a few classes it is created as a singleton. Component classes which are track, point and signal classes inherit their properties from a parent class called Element class. Element class contains common functions and variables of component classes. Component class instances are stored in an array for future use.

Secondly, the application software sorts out the information, which is received from the user interface. This sorting process will increase efficiency and lead to fast data processing. In order to carry out the sorting process, manager class creates an instance of Sort class and sends the component arrays to the Sort class using the constructer of Sort class. Two intermittent arrays are produced by this Sort class. The first one is called "PointInformation", which is composed of point class and contains point data such as where points are located on tracks. The other array is called "TrackInformation", which is composed of track class and contains information about all components that are located on the tracks and their locations. "PointInformation" array is used for producing of "TrackInformation" array by the Sort class, which is created by the manager class. Signal data are also needed as well as point data, where are stored in "PointInformation" array, in order to produce "TrackInformation" array. However, an array was not created for signals. The data, belonging to the signals in the station, are taken directly by manager class and transferred to the Sort class. Thereby, "TrackInformation" array is created based on "PointInformation" array and signal data. The algorithm that creates "TrackInformation" array is given in Fig. 6.

The third duty is to create an instance of CreateRoute class. CreateRoute class produces a final array which contains the number of possible routes and whose components are included in these routes and what the state of the components should be according to these routes. This is not an easy task as points can take two positions. Furthermore, multiple routes might be available while traveling from any entrance to any

```
Algorithm                                        Track Information Search
1:   for(i € tracks[])
2:       endpointx=tracks[i].locationx2
3:       endpointy=tracks[i].locationy2
4:       x=tracks[i].locationx1
5:       y=tracks[i].locationy1
6:       cont=true;
7:     while(cont)
8:       for(j € points[])
9:         if point[j].x1 is equal to x and point[j].y1 is equal to y then
10:              add point[j].id1 to track info
11:          elseif point[j].x2 is equal to x and point[j].y2 is equal to y then
12:              add point[j].id2 to track info
13:        endif
14:      endfor
15:      for(j € signals[])
16:        if signals[j].x is equal to x and signals [j].y is equal to y then
17:           add signals [j].id to track info
18:        endif
19:      endfor
20:      if endpointx is equal to x and endpointy is equal to y then
21:         add exit to track info
22:         cont=false
23:      endif
24:       decreament x
25:    end while
26:  endfor
```

Figure 6. Algorithm that creates “TrackInformation” matrix

```
Algorithm                                        Route Information Search
1:    for (i € Pointstates[,i])
2:      counter=0;
3:      position=0;//ray verilerindeki yer
4:      exit=true;
5:       while (exit)
6:         for (l € PointInformation[])
7:           if TrackInformation[position] is equal to pointInformation[l] and
8:             States[l,i] is equal to reverse
9:               then
10:                  add the point and its state to a temp
11:                  position=getnewposition()
12:                  currenttrack = getcurrenttrack()
13:                  increment position
14:              else TrackInformation[position] is equal to pointInformation[l] and
15:                   States[l,i] is equal to normal
16:              then
17:                  add the point and its state to a temp
18:                  increment position
19:          end if
20:          if TrackInformation[position] is equal to a signal
21:              add the signal to temp
22:              increment position
23:          end if
24:          if TrackInformation[position] is equal to a signal
25:              add the signal to temp
26:              increment position
27:          end if
28:          if TrackInformation[position] is equal to an exit then
29:              if TrackInformation[position] is equal to the desired exit then
30:                  add exit to temp
31:                  Routeinformation[,]=temp[,]
32:               else
33:                  exit = false;
34:              end if
35:              return "no route"
36:          end if
37:        end for
38:      end while
39:   end for
```

Figure 7. Algorithm that creates “RouteInformation” matrix

exit. Especially, in complex stations, where there are many points and routes, finding an optimal route inside all available routes becomes a tough task. In this study, in all possible routes, an optimal route, which is the shortest and simplest one, that is, it has the minimum number of points, is searched between an entrance and an exit in station topology. We created a matrix representing all possible positions of all points. Rows represent different combinations of states and columns represent points. Additionally, the optimal route must have minimum amount of points that are in diverging position and, because of this assumption, combinations of states are ordered inside the matrix by their number of reverse points. Once one of the rows of this matrix can get the train from the chosen entrance to the chosen exit points, it is assumed to be the optimal route because optimal routes have lower row number. The states of points along this route are stored along with the signals on the route inside an array called "RouteInformation". The algorithm written for this purpose is given in Fig. 7. After "RouteInformation" array is created, "XmlCreater" class is created and "RouteInformation" and "PointInformation" arrays are sent to its constructer.

Pattern net models can be produced using the data, which are stored inside Route Information and Point Information arrays. In an attempt to create pattern net models, first models are divided into blocks and each pattern net model contains an initialize block and the remaining blocks are associated with a component in a certain state.

The algorithm that produces track circuit models is given in Fig. 8. This figure is associated with Fig. 9, which contains track circuit pattern net model. The algorithm first selects an entrance point and creates pattern net model for this entrance by first initializing. Initializing is basically creating initial places, which are found in all route models, at the beginning of the algorithm. This process is shown in Fig. 9 as part A. Later each route that starts from that entrance is found and, for each point inside that route, a pattern net block is created. Part B in Fig. 9 represents a pattern net block for a point, which should be connected to the normal part of the previous point. Part D in Fig. 9 represents a pattern net block for a point, which should be connected to the reverse part of the previous point. Once all points are processed, the algorithm will reach an exit and for this exit another block will be placed on the respected pattern net model. Part C in Fig. 9 represents a pattern net block for an exit, which should be connected to the normal part of the previous point. Part C in Fig. 9 represents a pattern net block for an exit, which should be connected to the reverse part of the previous point.

```
Algorithm                                   Track Circuit Model
int PreviousPoint=null
string PlacedPoints[]=null
for(i ϵ RouteInformation[i,]) //i as respected route
    PreviousPoint=null
    PlacedPoints[]=null
    for(j ϵ RouteInformation[,j]) // j as components on the respective route
        if new route equals true then
            doinitilization() // see Figure9 Part A
        endif
        if RouteInformation[i,j] equal to a point then
            for(k ϵ PlacedPoints[])
                if PlacedPoints[k] equals to RouteInformation[i,j] then
                    PreviousPoint= RouteInformation[i,j]
                    break
                else
                    add point to PlacedPoints[]
                    if point is on normal then
                        place point on track circuit model // see Figure9 Part B
                        PreviousPoint = RouteInformation[i,j]
                    else
                        place point on track circuit model // see Figure9 Part D
                        PreviousPoint = RouteInformation[i,j]
                    endif
                endif
            endfor
        endif
        if RouteInformation[i,j] equal to an exit then
            if PreviousPoint is on normal then
                place exit on the track circuit // see Figure9 Part C
            else
                place exit on the track circuit // see Figure9 Part E
            endif
            break
        endif
    endfor
endfor
```

Figure 8. Algorithm that creates track circuit model

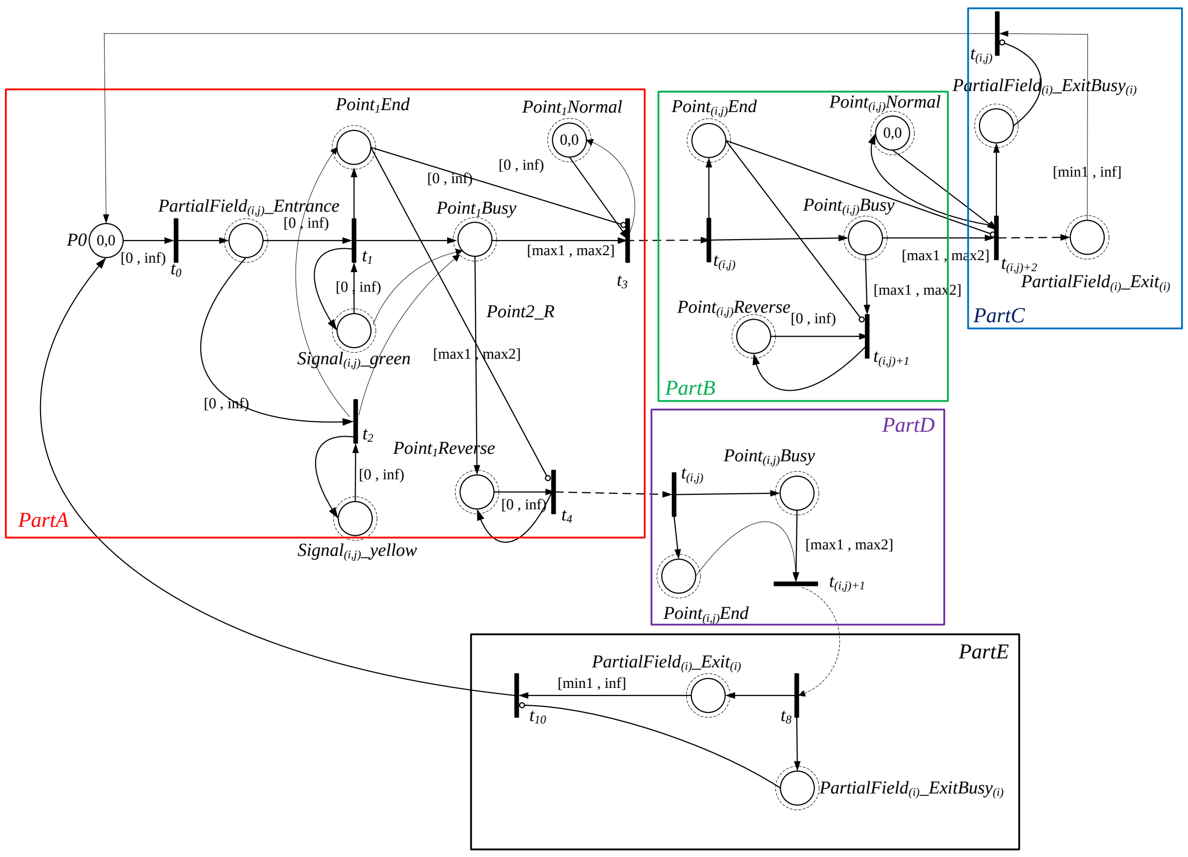

Figure 9. Sample pattern net blocks of track circuit models

```
Algorithm                                  Route Model
1:   int PreviousPoint=null
2:   for(i є RouteInformation[i,]) //i as respected route
3:   |  doinitilization() //see figure11 part A
4:   |  for(j є RouteInformation[j,]) //j as components on the respected route
5:   |  |  for(k є RouteInformation[k,])
6:   |  |  |  for(m є RouteInformation[m,])
7:   |  |  |  |  if RouteInformation[i,j] equal RouteInformation[k,m] then
8:   |  |  |  |     Place route as inhibitor on route model//see figure11 part B
9:   |  |  |  |  endif
10:  |  |  |  endfor
11:  |  |  endfor
12:  |  |  if RouteInformation[i,j] equal to a point then
13:  |  |  |  if point is on normal then
14:  |  |  |     place point on route model //see figure11 part C
15:  |  |  |     PreviousPoint = RouteInformation[i,j]
16:  |  |  |  else
17:  |  |  |     place point on routemodel //see figure11 part D
18:  |  |  |     PreviousPoint = RouteInformation[i,j]
19:  |  |  endif
20:  |  |  if RouteInformation[i,j] equal to an exit then
21:  |  |  |  place signal on route model //see figure11 part E
22:  |  |  |  break
23:  |  |  endif
24:  |  endfor
25:  endfor
```

Figure 10. Algorithm that creates route models

The algorithm that produces Route models is given in Fig. 10. This figure is associated with Fig. 11 which contains route pattern net model. The algorithm starts by selecting route from "RouteInformation" and creates a pattern net model for this route by first initializing as mentioned above. This process is shown in Fig. 11 as Part A. The algorithm searches for routes that have common components with the selected route and adds them to the model as an inhibitor as shown in Fig. 11 Part B. After, for each point on this selected route, a block pattern net model is produced and an inhibitor is added to the model to prevent the route

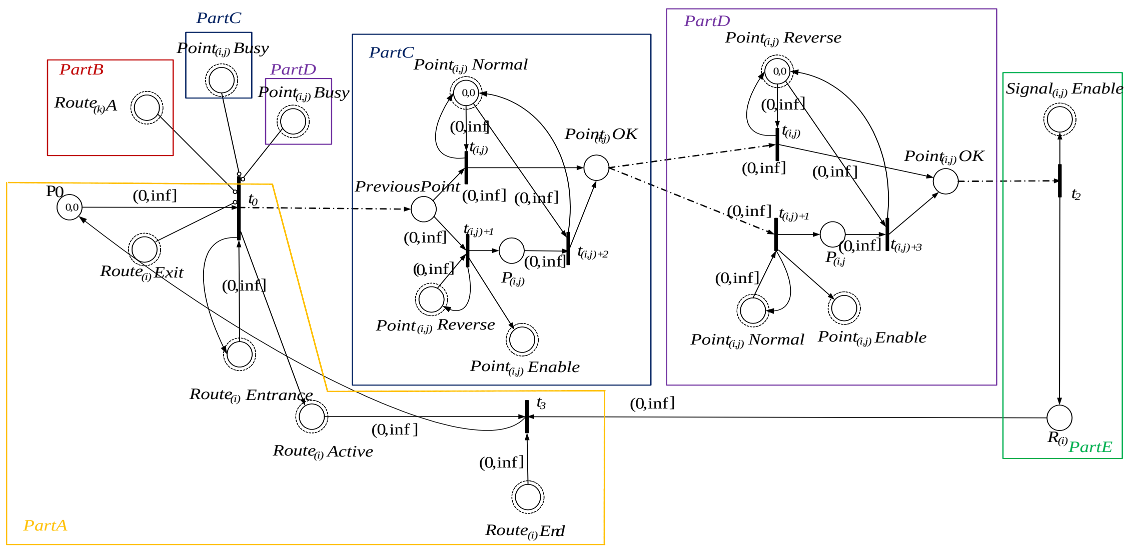

Figure 11. Sample pattern net blocks of route models

```
Algorithm                                   Signal Model
1:  bool reverse=false
2:  String ConnectedPoint=null
3:  for(i є signals[i])
4:    for(j є RouteInformation[j])
5:      if signals[i] equals RouteInformation[j,-] then
6:        doinitialize() // See Figure13 Part A
7:        for(k є routeinformation[,k])
8:          If (RouteInformation[j,k] equals the first point then
9:            ConnectedPoint= RouteInformation[j,k]
10:           Add connectedPoint to Signal Model // See Figure13 Part B
11:         endif
12:         if RouteInformation[j,k] point on reverse then
13:           reverse=true
14:         endif
15:       endfor
16:       if reverse equals true then
17:           add route on signal model // See Figure13 Part D
18:           reverse=false
19:         else
20:           add route on signal model // See Figure13 Part C
21:       endif
22:     endif
23:   endfor
24: endfor
```

Figure 12. Algorithm that creates signal models

from opening. Part C in Fig. 11 represents a pattern net block for a point which is normal position. Part D in Fig. 11 represents a pattern net block for a point which is in diverging position. Once all points are processed, the algorithm will need to enable the appropriate signal as shown in Fig. 11 Part E.

The algorithm that produces Signal models is given in Fig. 12. This figure is associated with Fig. 13 which contains a signal representation pattern net model. The algorithm starts by searching “RouteInformation” to find only the signals that are used to reduce complexity. Once a signal is found, an initialization block is produced as shown in Fig. 13 Part A. After all routes in “RouteInformation” that use the respected signal are searched for a reverse point. If there is no reverse point on the route, it is added to the model as an enabler for the green signal as shown in Fig. 13 Part C. Otherwise the route is added as an enabler for signal yellow as shown in Fig. 13 Part D. Once the point right after the signal is no longer busy, the signal should go back to red, so the respected point should be added as shown in Fig. 13 Part B.

4. Automatic Generation of Formal Models of Bastabya Station

Bastabya Station on T4 Topkapı-Habibler line operated by Istanbul Transportation Co. was chosen as a model. The station has five points, five signals and ten track circuits. Sets to represent the following items at Bastabya Station, whose topology is shown in Fig. 14, were defined: five points $P = \{p_1, p_2, p_3, p_4,$

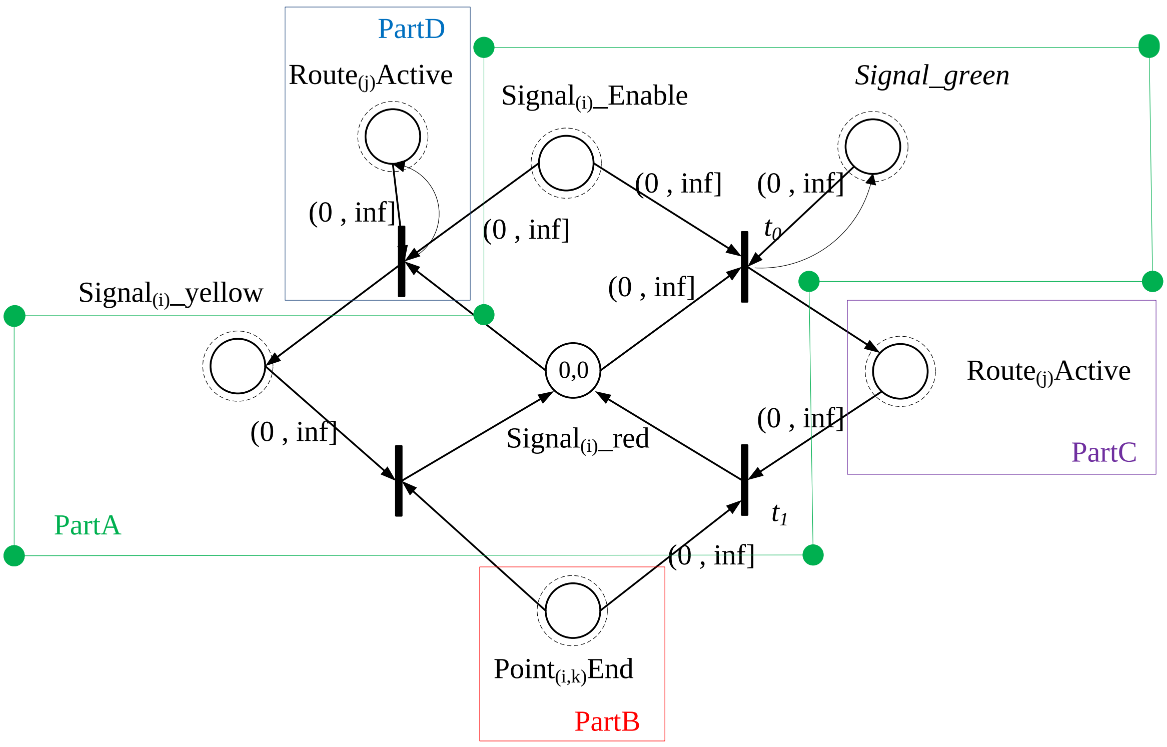

Figure 13. Sample pattern net blocks of signal models

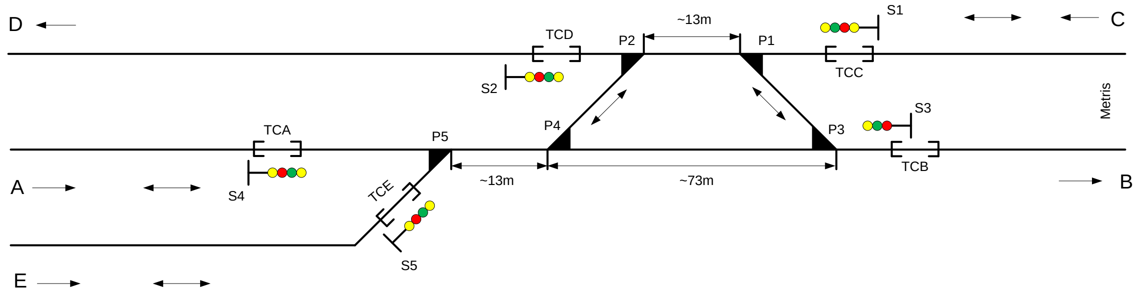

Figure 14. The topology of Bastabya station

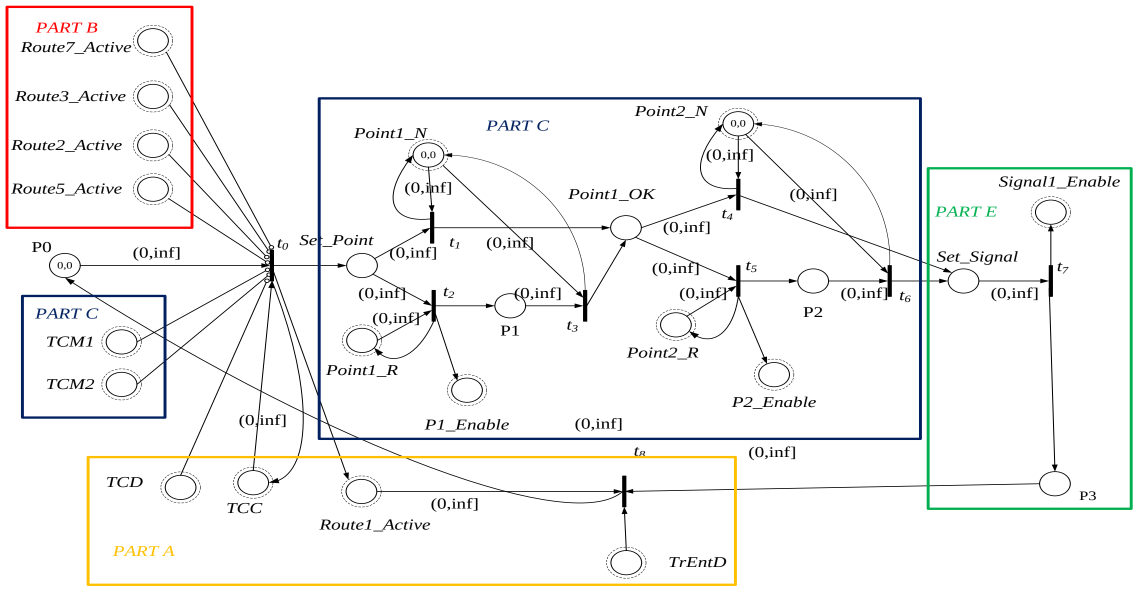

Figure 15. Timed-arc Petri net model of the route r_1

$p_5\}$, ten track circuits $TC = \{TCA,\ TCB,\ TCC,\ TCD,\ TCE,\ TCM1,\ TCM2,\ TCM3,\ TCM4,\ TCM5,\}$, first of five indicating the entering and departing of the station and the last five indicating the occupancy of the points as well as five signals $S = \{s_1,\ s_2,\ s_3,\ s_4,\ s_5\}$. The entrances of the station are identified as $TE = \{A, C, E\}$according to the operation of the station by Istanbul Transportation Co.

The routes identified can be opened for the trains on the condition that the track circuits are not occupied and the train proceeding on the second route to be opened should not be facing the train proceeding on the first route. Based on this, separate TAPN models were generated for each route through the developed software tool. As an example, the route r_1TAPN model, which is generated for a train proceeding on CD route, can be seen in Fig. 15.

Based on the model generated, route r_1 can be opened provided that TCC and TCD track circuits are unoccupied [PART A] and the relevant points (Point1 and Point2) are not occupied [PART C], either. In addition, any of the routes, which can be in conflict with r_1, should be opened; they should not be locked [PART B]. The points (Point1_N and Point2_N) on the route are placed in appropriate position in the right order once the route is chosen [PART C]. As a next step, signal 1 is enabled and green notification is transmitted [PART E]. At that point the train starts moving. Any route, which might clash with the route of the train, from C (the entrance point of the train into the station) to D (where the train leaves the station) is not allowed to be opened. The same situation applies for all the other routes. A new route can be opened on the condition that the track circuits on that route are unoccupied and the points are not occupied, either. It is also required that any other route has not been opened. Table 3 represents the points, their relevant positions based on the routes to be opened, and which track circuits are controlled according to the generated models.

Table 3. Track circuits, points and point positions by route

Entrance into the station	Route	Controlled Point and its Position	Track Circuit Controlled
C	r_1 (CD)	P1_N, P2_N	TCC, TCD TCM1, TCM2
	r_2 (CA)	P1_N, P2_R P4_R, P5_N	TCC, TCA TCM1, TCM2 TCM4, TCM5
	r_3 (CE)	P1_N, P2_R P4_R, P5_R	TCC, TCE TCM1, TCM2 TCM4, TCM5
A	r_4 (AB)	P3_N, P4_N P5_N	TCA, TCB TCM3, TCM4 TCM5
	r_5 (AC)	P1_N, P2_R P4_R, P5_N	TCA, TCC TCM1, TCM2 TCM4, TCM5
E	r_6 (EB)	P3_N, P4_N P5_R	TCE, TCB TCM3, TCM4 TCM5
	r_7 (EC)	P1_N, P2_R P4_R, P5_R	TCE, TCA TCM1, TCM2 TCM4, TCM5

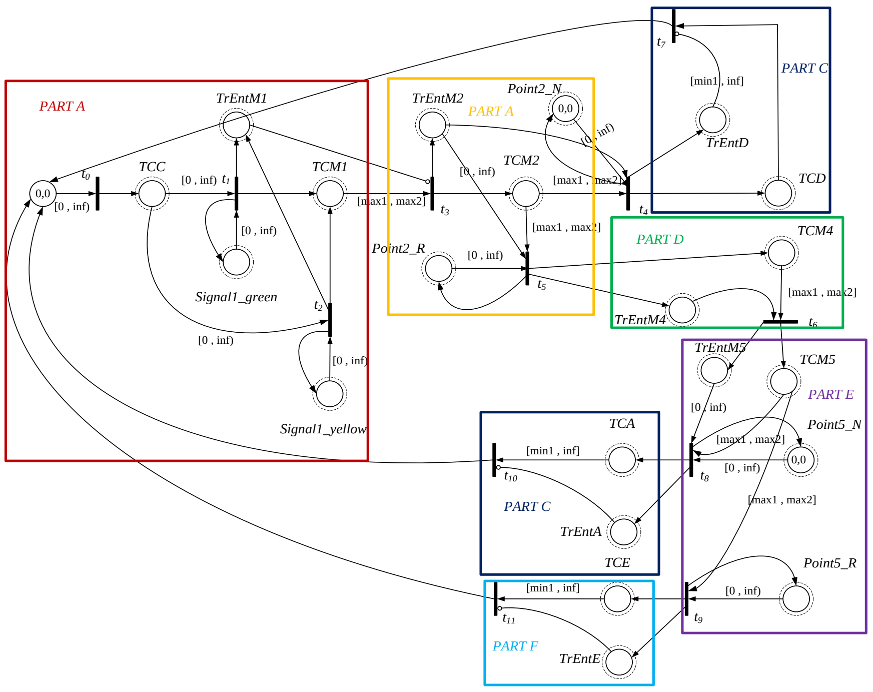

Figure 16. Track circuit timed-arc Petri net model for trains entering the station from C side

Track circuits working based on the occupancy principle, constantly provide the feed on where the trains are. This is a condition required for a safe journey. The set $\mathrm{TCx} = \{\mathrm{TCA}, \mathrm{TCC}, \mathrm{TCE}\}$ denotes the track circuit set occupied by the trains during their entrance into the Bastabya station whereas the set $\mathrm{TCy} = \{\mathrm{TCA}, \mathrm{TCB}, \mathrm{TCC}, \mathrm{TCD}, \mathrm{TCE}\}$ depicts the track circuit set occupied by the trains while they are leaving the station. With $\mathrm{x}, \mathrm{y} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$ and $x \neq y$, it is assumed that the train remains at the station as long as it does not pass from a second track circuit based on the route opened after it passes a track circuit. The automatic generated track circuit TAPN model, which indicates the actions of the trains entering the station from C side, can be seen in Fig. 16. Similarly, TAPN models for trains entering the station from A and E were also generated by the software tool.

As specified in the previous section and based on the model generated, the trains entering the station from C can leave the station from D, A or E depending on the route to be chosen. The train occupies the TCC track circuit initially. Then, it proceeds on the route opened, occupying one of the track circuits, which are TCD, TCA or TCE, and leaves the station.

5. Verification of Automatic Generated Formal Models for Point Automation System

It is of great importance to verify and prove that the automatic generated models for point automation system fulfill the identified safety requirements so that a safe journey can be ensured on railway systems. The safety requirements necessary for point automation system were identified in our previous study [5]. These safety requirements are added to the automatically generated models by the software tool. Thus, there is no need to write any query after the generation of the models. To verify the accuracy of the safety requirements (SR) identified in the point automation system, TAPAAL editor was used. The editor allows modeling, simulation and verification of the systems through TAPN. The verification of the identified safety requirements was made automatically as (EF, EG, AF, AG) was written on the CTL formulation, which is a subcategory of temporal logic. Thus, it is possible to determine whether the formulation verifies the generated model or not as a result of the verification procedure. All queries are checked via TAPAAL Discrete Verification method based on the Breadth First search order in state space. As the coverability tree is too large, it is not given in the study.

Safety requirements (safety requirement-SR), which are added to the automatically generated models by the software tool, can be listed as follows:

SR1: The point should either be in its normal position or in diverging position as it cannot remain in the same position concurrently.

SR1 is written in CTL formulation as $AG\neg(Point_k\_N \geq 1 \wedge Point_k\_R \geq 1)$. Table 4 represents the verification results and time for SR1.

Table 4. Verification results and time for SR1

Query	Result	Verification time
Point1_SR1	Satisfied	0.166 s
Point2_SR1	Satisfied	0.163 s
Point3_SR1	Satisfied	0.163 s
Point4_SR1	Satisfied	0.165 s
Point5_SR1	Satisfied	0.165 s

SR2: For a point to be locked, the point should either be in its normal position or in diverging position.

SR2 is written in CTL formulation as $AG\neg(P_k\_Enable = 0 \wedge (Point_k\_N \geq 1 \wedge Point_k\_R \geq 1))$. Table 5 represents the verification results and time for SR2.

Table 5. Verification results and time for SR2

Query	Result	Verification time
Point1_SR2	Satisfied	0.164 s
Point2_SR2	Satisfied	0.165 s
Point3_SR2	Satisfied	0.166 s
Point4_SR2	Satisfied	0.165 s
Point5_SR2	Satisfied	0.169 s

SR3: The point should not be moving while the train occupies any point, which means while the train is on its way over the point, it should not get any point engine command or move.

SR3 is written in CTL formulation as $AG\neg(TCM_k \geq 1 \wedge (P_k.NtoR \geq 1 \vee P_k.RtoN \geq 1))$. Table 6 represents the verification results and time for SR3.

Table 6. Verification results and time for SR3

Query	Result	Verification time
Point1_SR3	Satisfied	0.166 s
Point2_SR3	Satisfied	0.167 s
Point3_SR3	Satisfied	0.166 s
Point4_SR3	Satisfied	0.170 s
Point5_SR3	Satisfied	0.165 s

SR4: Signal should be locked into green, yellow and red light, referring to normal direction, siding direction and stopping direction, respectively. The train should start moving when the signal notifies proceeding direction, and the signal should give red notification again once the train occupies the first track circuit.

SR4 includes two different requirements to be written in the CTL formulation. They are written as $AG(Signal1\_green \geq 1 \vee Signal1\_yellow \geq 1 \vee Signal1\_red \geq 1))$ and $AG\neg(Signal1\_green \geq 1 \wedge (RCM1 \geq 1 \wedge RCM2 \geq 1))$.

SR5: When the route selected is locked and opened, the points on the route should also be locked in the relevant position and there should be no proceeding until the route is free.

SR5 also includes two different requirements to be written in the CTL formulation. They are written as $AG\neg(Route1locked \geq 1 \ \wedge (Point1\_R \geq 1 \ \vee$ $Point2\_R \geq 1))$ and $EF(Route1locked \geq 1 \ \wedge$ $(Point1\_N \geq 1 \ \ \wedge Point2\_N \geq 1))$.

SR6: Points should firstly be locked based on the route chosen. Then, relevant signal notification should be given when the route is locked.

SR6 is written in CTL formulation as $AG\neg(Route1locked = 0 \wedge Signal1\_green \geq 1)$. Table 7 represents the verification results and time for SR4, SR5 and SR6.

Table 7. Verification results and time for SR4, SR5 and SR6

Query	Result	Verification time
Signal1_SR4_1	Satisfied	0.166 s
Signal1_SR4_2	Satisfied	0.167 s
RouteCD_SR5_1	Satisfied	0.166 s
RouteCD_SR5_2	Satisfied	0.170 s
RouteCD_SR6_1	Satisfied	0.170 s
RouteCD_SR6_2	Satisfied	0.165 s

6. Conclusion

A software tool, which can be used for automatic TAPN model generation from the station topology and verification of the generated models for point automation system, was successfully developed. TAPN models of Bastabya station, operated by Istanbul Transportation Co. in Turkey, were formed automatically by using the developed software tool based on CENELEC EN 50128. Additionally, it was verified and proven through temporal logic, one of the formal methods recommended by CENELEC EN 50128 standard, that the generated TAPN models fulfilled the identified safety requirements. The tool was tried for different stations on M1 Aksaray-Airport metro line, operated by Istanbul Transportation Co. and successful models were obtained. The TAPN models, which are generated for these stations, were also verified. As a result, the information, which is shown in Table 8, was obtained.

It was concluded that the tool significantly reduced the modeling faults caused by human factor and improved the efficiency in the generation of the models. Moreover, reliability and safety of the whole system increases thanks to automatic generation of the system models. Our future work will focus on the development of a process to generate the discussed models automatically from the programming codes, which are written in STL format in Programmable Logic controller.

Table 8. Results of model generation and verification for different stations

Station	Number of				Application Runtime	Verification	
	Points	Tracks	Entrances	queries	Second(s)	Mb	Second(s)
Aksaray	2	2	4	48	0.724	84	6.662
Davutpaşa	2	2	2	34	0.654	48	3.525
Esenler	1	2	2	26	0.642	33	2.819
Otogar	4	3	4	78	1.007	428	615.815
Bahçelievler	2	2	4	48	0.755	150	22.2
System Info	Intel Core I7-2630QM Cpu @2,00 Ghz x64 6 Gb Ram Windows 8 64-Bit						
Verification tool	Tapaal 3.00						

Acknowledgment

This work is supported by Yıldız Technical University Research Project Fund under Grant 2012-04-04-KAP04.

References

[1] **CENELEC EN 50128.** Railway applications - Communication, Signaling and Processing Systems - Software for Railway Control and Protection Systems, 2011.

[2] **N. A. Zafar.** Formal specification and validation of railway network components using Z notation. *IET Software,* 2009, Vol. 3, No. 4, 312-320.

[3] **K. Winter.** Model checking railway interlocking systems. *Australian Computer Science Communications,* 2002, Vol. 24, No. 1, 303-310.

[4] **M. Banci, A. Fantechi.** Geographical Versus Functional Modeling by Statecharts of Interlocking Systems. *Electronic Notes in Theoretical Computer Science,* 2005, Vol. 133, 3-19.

[5] **I. Sener, O. T. Kaymakçı, I. Ustoğlu, G. Cansever.** Specification and formal verification of safety properties in point automation system. *Turkish Journal of Electrical Engineering and Computer Sciences, 2014* (Accepted). DOI: 10.3906/elk-1311-27.

[6] **L. Jacobsen, M. Jacobsen, M. H. Moller, J. Srba.** Verification of timed-arc Petri nets. *Lecture Notes in Computer Science*, 2011, Vol. 6543, 46-72.

[7] **J. A. Mateo, J. Srba, M. G. Sørensen.** Soundness of timed-arc workflow nets. *Lecture Notes in Computer Science*, 2014, Vol. 8489, 51-70.

[8] **M. Andersen, H. G. Larsen, J. Srba, M. G. Sørensen, J. H. Taankvist.** Verification of liveness properties on closed timed-arc Petri nets. *Lecture Notes in Computer Science*, 2013, Vol. 7721, 69-81.

[9] **S. V. Birch, T. S. Jacobsen, J. J. Jensen, C. Moesgaard, N. N. Samuelsen, J. Srba.** Interval abstraction refinement for model checking of timed-arc Petri nets. *Lecture Notes in Computer Science,* 2014, Vol. 8711, 237-251.

[10] **A. Kuzu, O. Songuler, A. Sonat, S. Turk, B. Birol, E. H. Dogruguven.** Automatic interlocking table generation from railway topology. In: *Proceedings of the IEEE International Conference on Mechatronics,* Istanbul, Turkey, April 13-15, 2011, pp. 13-15.

[11] **U. Yildirim, M. S. Durmus, M. T. Soylemez.** Automatic interlocking table generation for railway stations using symbolic algebra. In: *Proceedings of the 13th IFAC Symposium on Control in Transportation Systems,* Sofia, Bulgaria, September 12-14, 2012, Vol. 13, pp. 171-176.

[12] **Y. Cao, T. Xu, T. Tang, H. Wang, L. Zhao.** Automatic Generation and Verification of Interlocking Tables Based on Domain Specific Language for Computer Based Interlocking Systems (DSL-CBI). In: *Proceedings of the IEEE International Conference on Computer Science and Automation Engineering*, Shanghai, China, June, 10-12, 2011, Vol. 2, pp. 511-515.

[13] **M. S. Sachdev, P. Dhakal, T. S. Sidhu.** A computer-aided technique for generating substation interlocking schemes. *IEEE Transactions on Power Delivery*, 2000, Vol. 15, No. 2, 538-544.

[14] **X. Chen, Y. He, H. Huang.** A component-based topology model for railway interlocking systems. *Mathematics and Computers in Simulation,* 2011, Vol. 81, No. 9, 1892-1900.

[15] **Z. Yong, P. Zhibin, Y. Chungui.** Research on the method and tool for automatic generation of application data for interlocking simulation system. *International Conference on Control, Automation and Systems Engineering,* Singapore, July 30-31, 2011, pp. 1-4.

[16] **A. E. Haxthausen.** Automated generation of safety requirements from railway interlocking tables. *Lecture Notes in Computer Science,* 2012, Vol. 7610, 261-275.

[17] **A. David, L. Jacobsen, M. Jacobsen, K. Y. Jørgensen, M. H. Møller, J. Srba.** TAPAAL 2.0: integrated development environment for timed-arc Petri nets. *Lecture Notes in Computer Science,* 2012, Vol. 7214, 492-497.

Received June 2014.