Metrol. Meas. Syst., Vol. 24 (2017), No. 3, pp. 473–488.

PAN POLSKA AKADEMIA NAUK

METROLOGY AND MEASUREMENT SYSTEMS

Index 330930, ISSN 0860-8229

www.metrology.pg.gda.pl

M&MS

ANALYSIS OF ECONOMICAL LIGHTING OF HIGHWAYS IN THE ENVIRONMENT OF SMOL LANGUAGE

Zdzisław Kowalczuk, Jakub Wszołek

Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics, G. Narutowicza 11/12, 80-233 Gdańsk, Poland (✉ kova@pg.gda.pl, +48 58 347 2018, jwszolek@eti.pg.gda.pl)

Abstract

The paper puts forward and implements a method of designing and creating a modelling simulation environment for extensive and complete analysis of economical lighting on highways. From a general design viewpoint, the proposed solution explores the concept of a network description language (SMOL), which has been designed to describe the necessary network functions, mechanisms, and devices for the purpose of their computer simulation and verification. The presented results of the performed research confirm the usability of intelligent lighting on highways, both in the sense of the design concept and in the aspect of saving energy.

Keywords: lighting systems, economical factors, engineering systems, monitoring, domain language, modelling and simulation environment.

1. Introduction

Economic issues encourage us to look for new opportunities of better solving specific technical problems. Thanks to knowledge, technology, and experience we can obtain higher optimality (reliability and effectiveness) of the designed systems.

The modern automatic control systems are to a great extent founded on measurement networks. Current monitoring gives us the possibility of verifying the correctness of a system's operation. The observed dynamic development of technology causes that computer-based automatic control systems have growing possibilities of analysing and using measurement data (and often saving them in a central place). The current techniques of data analysis (data mining – a data exploration process) enable us to develop useful mechanisms of detection or prediction of failures on the basis of symptoms hidden in the measurement data. Failure of a large management or automatic control system usually means a complete or partial stopping of the process (and financial losses). Therefore, tools for modelling and simulating such systems are significant in the analysis aiming at identification of weak points of the developed control solutions.

Within the research, a system [7–10] was developed in order to enable description of network structures of interconnected devices in SMOL (a language (…L) for describing networked *Systems for Monitoring Objects* (SMO); an original, dedicated domain language, and to run appropriate simulations.

Our study is a direct response to the research needs of one of Polish companies, specializing in the production of electronics. Since the presented model of SMO can easily map specific conditions of the measurement environment, we can consider all most essential cases and examine them in terms of cost-effectiveness of the intended intelligent system for measuring and controlling lighting on highways.

To perform this kind of detailed simulation studies, the SMO project has been expanded to

Article history: received on Oct. 10, 2016; accepted on Mar. 28, 2017; available online on Sep. 01, 2017; DOI: 10.1515/mms-2017-0041.

its branch of the description programming language SMOL (suitable also for implemental issues), and next to the complete system SMOLsim of modelling and simulation.

This paper briefly presents the concept of the language and a practical example demonstrating the possibilities of the developed simulation environment. Technical details related to the way of using this system in relation to the requirements of the considered application, are also included.

2. Modelling network structures using SMOL language

Development of measurement techniques resulted in the formation of a new branch of distributed measurement, which is currently used to engineer intelligent management and control of objects. The dynamic development of this branch of science was also an important incentive for the authors to develop an original mechanism for modelling structures for *measurement-diagnostics-control* (MDC) and the related MDC computer networks discussed here. In effect, in this paper, we first focus our attention on a method of describing the MDC networks in the dedicated language, SMOL. Proper description of a network structure, by means of SMOL, lays down the foundations for further analysis of the network and its optimisation.

2.1. Language concept

SMOL, a dedicated language of semantics enabling the designer to model measurement-diagnostics-control networks, is an example of a domain-specific tool, tailored to solve problems from a specific area [1].

Description of a network structure is an essential basis for representing the problems of configuration, re-structuring, and optimisation of network subsystems. In other words, SMOL is used to describe the components of a diagnostic network, and the accompanying connections, along with their appropriate parameterisation. Such a formal representation will also enable to verify the future process of implementing such a network with the use of a relevant parser, which is a software platform for analysing a programmed network structure and its parameters.

The *semantics of the language* (SMOL) are based on structures of hierarchical processing. It is assumed that all network signals are eventually bundled together in the Central Node, which is a central vertex of the analysed graph.

2.2. Components of network description

One of the best ways of modelling industrial computer networks is to use a directed flow graph as a tool suitable for describing relations occurring in a network. Such a model is represented by a triple of sets (vertices, edges, and mappings). By defining these sets, we can precisely present an information flowchart in MDC networks.

Below, we introduce the basic types of nodes used in graphs to properly represent network relations. The main division concerns the use of static and dynamic nodes implementing transforming functions [10].

2.3. Central node

In general, the *central node* (CN component) is an object of MIMO (*Multiple-Input Muliple-Output*) type (with multiple inputs and outputs). It is a necessary component in defining practical MDC networks. In an actual computer connection network, such a component represents the centre for transmitted information that receives data from measuring gauges and

transmits information or control signals to receivers or actuators. A general scheme of this node is shown in Fig. 1.

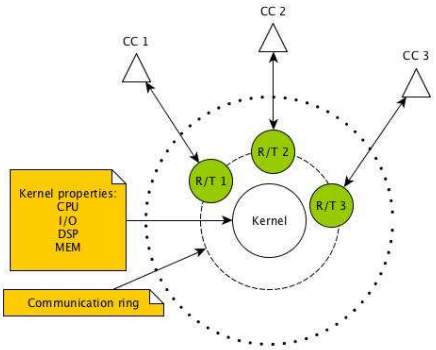

Fig. 1. Central node representing a multidimensional (MIMO) object.

The central node has a set of *communication channels* (CC) and associated *Receiver/Transmitter* (R/T) modules, belonging to the communication layer referred to as the Communication Ring, used to exchange information between the Kernel and other nodes. The kernel of CN is implemented as a computer system defined through proper hardware resources and their parameters (CPU – *Central Processing Unit*, I/O *Input/Output*, DSP – *Digital Signal Processing* or MEM – *Memory*).

The central node is always required when we intend to practically verify feasibility of implementing a designed network.

2.4. Transferring node

Transferring node is a universal element of MIMO type, which functions as a transmitter (TN element) in the network; it has a structure shown in Fig. 2, where CC and R/T components function analogously to those described above for the central node.

The kernel of this node can perform all necessary T(**A**) functions transforming information **A**. The concept of using such "transformers" is presented in Subsection 2.6.

In an actual implementation of an MDC network, the TN components have specific inputs and outputs, and the desired methods of signal transmission assigned to them. In practice, transmitters are working in most universal contemporary wire and wireless standards (WiFi – *Wireless Fidelity*, ZigBee – IEEE *802.15 standard*, Switch – *http://en.wikipedia.org/wiki/Network_switch*, Modem).

2.5. Expander

In the real-world solutions, data buses are commonly applied as components specifically dedicated to transmitting measurement and control data. Therefore, such a function is implemented in the SMOL language intended to be a tool for designing industrial MDC network systems.

Expander, representing a specific transmission component of designed network, gives the possibility of exchanging information on the basis of buses (PROFIBUS – *https://en.wikipedia.org/wiki/Profibus*, CAN – *https://en.wikipedia.org/wiki/CAN_bus*, RS-232, RS-485, DSB – *Diagnostic Service Bus* and KNX/EIB – *https://en.wikipedia.org/wiki/KNX_(standard)*) commonly used in industrial measurement and automatic control

systems. The type of the applied protocol is pre-defined by means of a list of the parameters available for this (Expander) object.

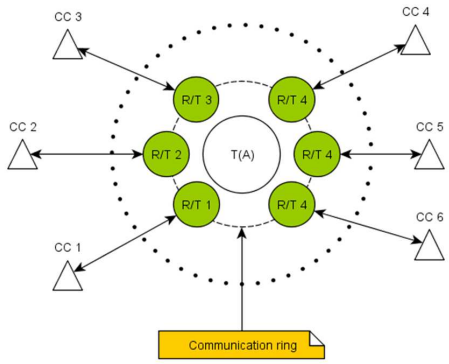

Fig. 2. Transferring node: a universal MIMO node.

2.6. Transformer – transforming function

MDC systems often require quick, and rather simple, modification of the transmitted data stream. Therefore, *transformers* (TRs) enable to implement a program of universal mathematical T(**A**) function (sum, difference, product, quotient, differentiation, integration, averaging, and some slightly more complex digital filters) operating on the input data. Currently, the possibility of using transformers is provided only for CN and TN nodes, described in Subsections 2.3 and 2.4.

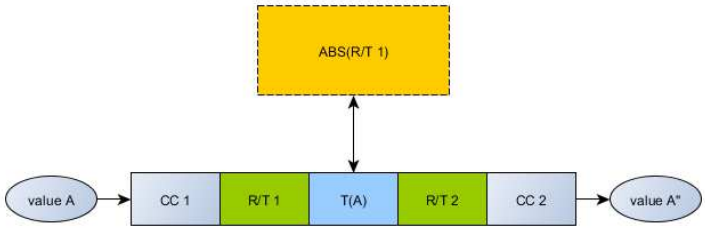

Fig. 3. TN node with a transforming function T**(A)** implemented as TR.

The mechanism of transformers' functioning comes down to operations on data **A** that flow into the node through the input communication port (CC_1) and are transferred outside through the output communication port (CC_2). Fig. 3 shows an approximate working diagram of a model transformer receiving data **A** through port CC_1, modifying them according to a selected function T(**A**) = A* **(**performing, for instance, a simple function T(**A**) = abs(**A**) = |**A**|), and using port CC_2 for displaying the results **A*** through.

2.7. Parameters of connections between nodes

By defining the parameters of direct connections between the system's nodes, that is, the parameters of *communication channels* (CC), one specifies the character of a medium used for transmission (*e.g.* Ethernet network). The designer can determine the necessary physical parameters of considered communication channel; for example, a length of physical medium,

a type of it (air, copper cable, computer network, *etc*.), attenuation, and an estimated maximum data transfer speed.

2.8. Sensor/actuator node

It is a common type of node terminating the network structure in particular branches. In actual MDC networks this component usually represents a sensor and a measuring gauge, or display a gauge and (logical) states of control actuators.

2.9. Simulation environment

The concept of SMOL language has been based on solutions meeting the requirements of the object-oriented programming paradigm (namely, abstraction, polymorphism, encapsulation, and inheritance) [8]. This approach facilitates mapping of pertinent relations occurring in actual MDC systems.

A program written in the SMOL language describes a set of processes/objects communicating with each other in order to execute specific tasks. The proposed different types of nodes [10] enable us to implement objects having functions corresponding with devices commonly used in communications, diagnostics and automatic control engineering. These objects – represented according to the high-level programming language concept – have a structure defined by means of variable fields and behaviours. They are methods or functions embedded in the object (a function occurring independently in the code is considered to be a regular function). This enables the programmer to unequivocally define the roles performed by objects in the program, and organize the code structure. When programming in SMOL, the designer modelling a network structure does not have to limit himself to the use of the provided/implemented ready objects. Due to the application of open architecture, it is possible to add (with the use of the inheritance mechanisms and relevant interfaces) customized behaviours of the designed components of an MDC system. All this contributes to the ease of development of a SMOL environment, and enhances its universality.

As it has been mentioned, a SMOL environment contains many packages of ready objects and functions. To facilitate the designer's work, the objects are divided into two main groups.

SMOH is a group of objects referring to hardware components. The user can therefore employ ready-to-use objects, which are most frequently used in MDC networks. Examples of such objects are: a network switch, gauge, sensor or actuator. Each of the ready components has all mechanisms enabling to connect it with other network components. The connection of objects is performed through a specially defined interface (obviously, the objects which have no interface defined cannot be connected to other objects requiring an interface for communication). On the basis of the provided basic set of components and functions, the designer can customize objects with the use of the inheritance mechanism.

SMOF is a set of advanced functions useful in implementation of data flow and performing processes in MDC networks. Transforming functions are included in the system package. The SMOF package, among others, includes an implemented Dijkstra's algorithm for finding the shortest path between network nodes, and an Edmonds-Karp algorithm for computing the maximum flow either in the whole network or in a specific part of the network. Moreover, the SMOF package gives the possibility of generating a graphic representation of the modelled computer network environment. Displaying the structure of connections gives rise to better understanding of the network; the relations between its particular components have an impact on the final evaluation of the network's applicability, and facilitates network optimisation.

2.10. Simulation system architecture – SMOL platform

Once the language was developed, which enabled description of the network's connected measurement and automation devices, we began designing a corresponding tool for simulating the developed structures.

The intended SMOL simulation platform is composed of several interconnected modules. Each of the modules performs a specific function. The objective of implementation task was to develop a possibly general architecture enabling simple modification, redevelopment and integration of designed MDC networks.

The structure of software components included in this application is shown in Fig. 4.

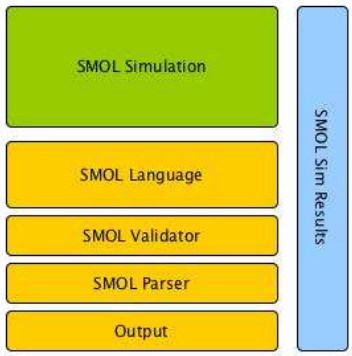

Fig. 4. A diagram of the SMOL architecture solution.

The SMOL platform is based on the Groovy technology (an object-oriented scripting language patterned on Java syntax, enriched with additional features such as closures; *http://www.groovy-lang.org/*) [2–3]. It is an object-oriented scripting language patterned on the Java syntax and run on a *Java Virtual Machine* (JVM) [4]. Due to its dynamic typing, closures, operator overloading and support for meta-programming (meta-classes, categories, AST (*https://sewiki.iai.uni-bonn.de/research/jtransformer/start*) transformations), SMOL is well suited for building customized domain-specific extensions. The applied technology enables simple integration of the languages, Groovy and SMOL. There is also the possibility of interlacing both languages, in order to achieve greater flexibility in programming using the SMOL platform. The system's designer can even implement a part of the functionalities developed in Java, and then embed it – through Groovy – directly in the SMOL language.

Such system functionality significantly enhances the capabilities of both the expansion and analysis of the discussed models of MDC networks. An unquestionable advantage of this solution is its openness to various modifications, enabling to use the environment in diverse industrial, engineering, and scientific applications, as well as in student projects.

2.11. SMOLsim codebase

The discrete simulation environment SMOLsim was implemented by using the Java and Python technologies. In the design work we aimed at developing a possibly universal software solution, which gives the ability to promptly implement different simulation scenarios and presents a great flexibility in the usage. Certainly, an important objective of the simulation module was also its full compatibility with the SMOL language.

In the developed simulation environment, we use discrete functions both in the formal description (characteristics of the state variables of the system) and in the description

of dynamics (measurement of the elapsed time). Accordingly, in this event-driven simulation, changes (events) occur only at specific, discrete moments. This type of simulation can be implemented based on two methods: 1) planning events and 2) process interaction [14].

According to the design assumptions, a network description expressed in the SMOL language is transformed to a form compatible with the SMOLsim platform. The conversion performed using scripts written in the Python environment, yields a simulation project coded in the Java language. Thus, this project is also consistent with a commonly used tool – Apache Maven (*https://maven.apache.org/*), which automatizes building software for the Java platforms.

During the work on the simulation environment SMOLsim, we used common open-source third-party libraries. Such a solution enhances the versatility and flexibility of the environment. In addition, the generated Java code gives the opportunity to introduce modifications in the object's behaviour, what also extends the functionality of the simulator. A sample of Java code generated for one of the simulated objects (defined as a part of the description of the network of intelligent lamps modelled in SMOL) is attached in Appendix 3.

2.12. Examples of SMOL platform applications

In recent years, the development of smart buildings increased the possibilities of using the platform (SMOL) for modelling and simulating MDC networks. Smart buildings are currently a synonym for a productive and economically efficient environment, which enables to optimise various components: systems, structures, services, management, as well as the internal relations between them [16].

More and more complex systems of optimal management of heating systems, in buildings using passive heating and cooling, are nowadays being widely developed [5, 6]. Moreover, there are hybrid energy systems using renewable energy sources. Smart buildings also involve monitoring and alarm systems, fire systems and access control systems. These solutions use a common network for communication between particular modules. The leading aim of the work on the SMOL platform was the development of a tool for simulation and optimisation of operation of implemented MDC networks [9], with a particular emphasis on object-oriented and building automation systems.

Thus, apart from flexibly creating a description of general MDC structures and simulating them, the SMOL platform also offers the possibility of describing a structure of measurement, diagnostic and automation functionalities used in buildings, by means of a special domain-specific programming language. An unequivocal description of the network structure is the necessary basis for precise simulation and effective optimisation of the developed networks.

Recall that the architecture of SMOL, based on interfaces and abstract classes, enables to easily expand the functionalities. Also, the implemented basic classes open the possibility of creating customized node implementations using the provided methods of communication between objects and other libraries for free expansion of SMOL. An example of such expansion is the use of SMOL for simulating BAS (*Building Automation System*) and BMS (*Building Management System*) systems along with an expander simulating exchange of information between the devices by means of KNX/EIB bus (or other type). The communication media can be a twisted pair of cables, Ethernet network, energy cables, radio or infrared waves. The central part of the network is a line, which can have 64 bus devices connected to it. The information is sent asynchronously at a speed of 9600 bps [11–13].

KNX/EIB bus is one of the most frequently chosen solutions in implementing building automation systems.

The SMOL tool, matched for simulation of such building systems, suitably maps the specific mechanisms used in communication between building automation devices.

An example of creating an algorithm with the use of SMOL for modelling and simulating is presented in Figs. 5 and 6. The former shows the concept of a measurement network using a KNX/EIB interface for communication purposes, and the latter presents this network in a graphic form using the SMOL language, as a right solution for network representation, detailed modelling, and target optimisation.

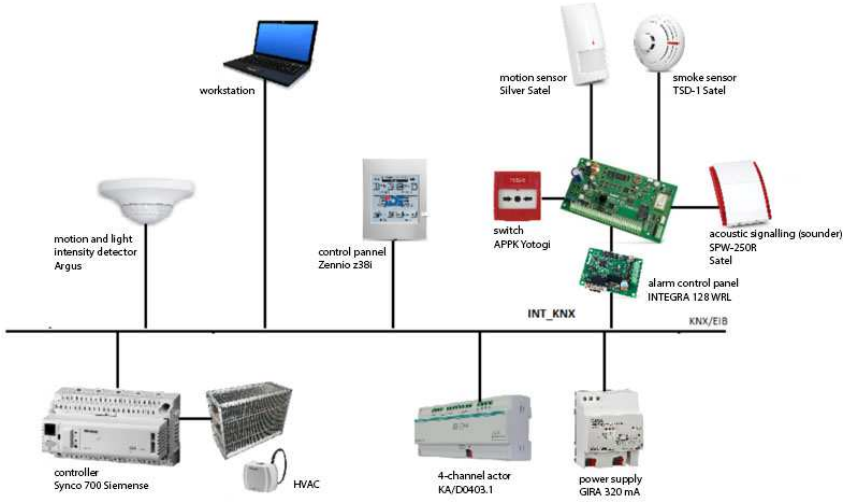

Fig. 5. The concept of a measurement network based on a KNX/EIB interface.

The software code in SMOL language describing the structure of presented MDC network is included in Appendix 1.

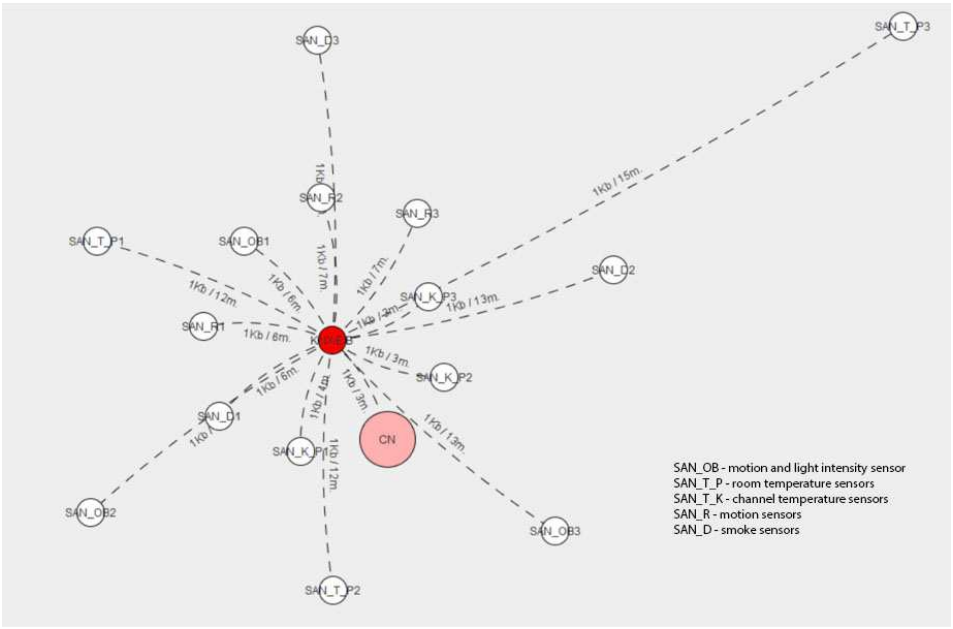

Fig. 6. A graphic representation of KNX/EIB network created with the use of SMOL environment.

3. Using SMOL environment in designing smart lighting

Let us now consider in a more detailed way an example, illustrating the working principle of the SMOL modelling and simulation system for distributed MDC network systems, and representing the concept of a road lighting system. In particular, in this section we present a practical application of the platform, SMOL, concerning the development and verification of a certain method of economical lighting of a highway. Below, we describe the subject of analysis, the simulation method, and the research results along with an assessment of suitability of the offered solution.

Figure 7 shows the resulting graph of the analysed network of the system, managing highway lighting, generated by means of a SMOL parser. The corresponding software code written in SMOL describing the structure of this MDC network is given in Appendix 2.

3.1. Object of analysis

In the analysed example shown in Fig. 7, as programmed and exercised in SMOL, the lamps communicate with each other and exchange information about detection of a moving vehicle.

The ultimate aim of this analysis is finding an answer to the question whether – from the viewpoint of possible savings – the designed MDC system is worth using for automatic management of road lighting. The presented test scenarios should also show the possibilities of effective implementation of different MDC lighting systems and (potential) verification of energy profitability of the offered solutions.

A situation diagram being the basis for scenarios of saving energy of the road lighting, as discussed here, is shown in Fig. 8.

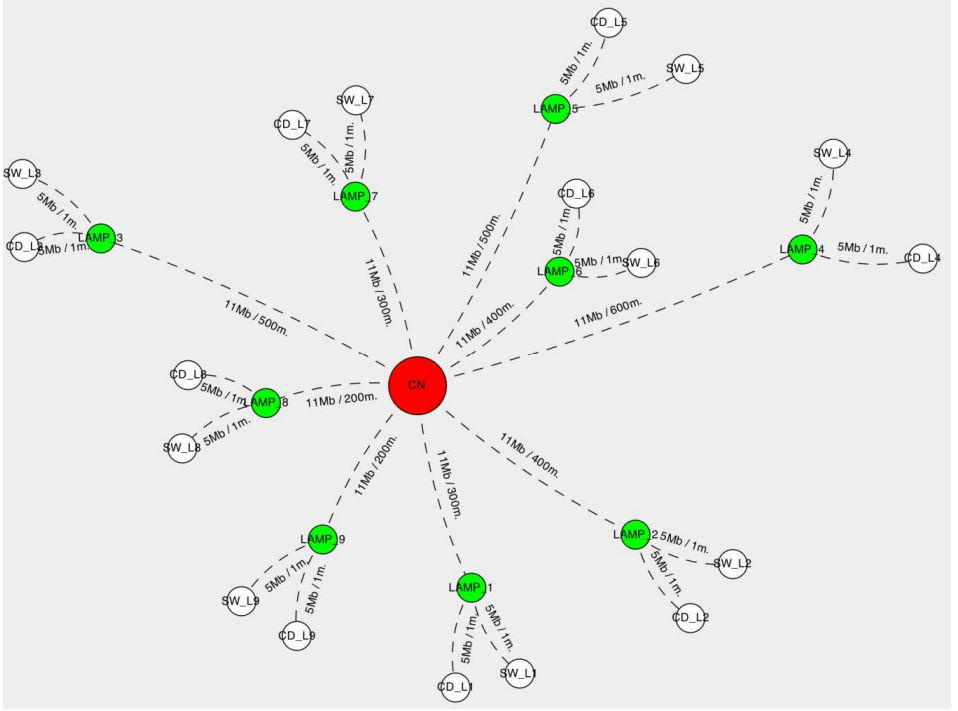

Fig. 7. A graphic representation of the network structure used in managing road lighting obtained as a result of SMOL parsing.

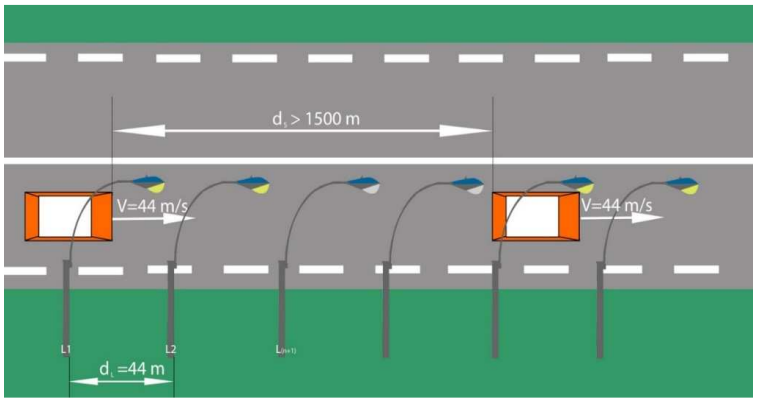

Fig. 8. A situation diagram of the analysed road simulation.

3.2. Research scenarios

We are going to present here two test scenarios. Through simulated testing of the designed network, we can spot and identify advantages and disadvantages of the discussed solution.

Before starting the experiments, let us make assumptions specifying simple and rational scenarios of simulations. Thus, we assume that all cars drive keeping the same distance from each other, at a constant speed of 160 km/h (44 m/s), and that the distance between each two successive road lamps is 44 m, and the period of a light's switching-on is 2 sec. The minimum period of a lamp being on is defined as 10 sec., and the period between switching-off and switching-on again is 5 sec. The technological requirements for the lamp operation cycle are explained in Fig. 9. Moreover, we determine how the lamps are switched on in relation to a moving car when a car is detected: a "current" lamp, which is the one whose sensor has just detected the presence of a car, is on and, by assumption, three lamps in the direction of the car movement are also on, which is shown in Fig. 10. For the safety of driving at a fixed speed of 160 km/h, we further assume that entering the road invokes initially switching on four lamps (obviously, one can use here also an adaptive algorithm, which adjusts the number of lights to the car's speed) at once.

The resulting diagram representing the hardware configuration of the analysed solution is shown in Fig. 7. In this study we consider the first 9 road lamps fitted with a sensor detecting a moving car (*CD_n – Car Detection*) and actuators responsible for switching lights on and off (*SW_n – Switch*).

1. **Off-wave OfW (switching off)**. The first experiment represents a situation where cars are driving in the same direction (one after another) at the same speed. Distances between cars are greater than 1500 metres. During this test we expected to observe the effect of automatic switching-off of the lights. Due to the lack of information about a new approaching car from the sensors, the lamps are switched-off, and lighting keeps up only to each single car, currently moving.
2. **On-wave OnW (switching on)**. The aim of this research scenario is to analyse a situation where cars are driving in the same direction (one after another) at the same speed, with a fixed distance between them, being smaller than 440 metres (10 × 44 m, the distance between the lighting posts). Due to this extremely high traffic volume, we expect the effect of constant lighting of the road lamps.

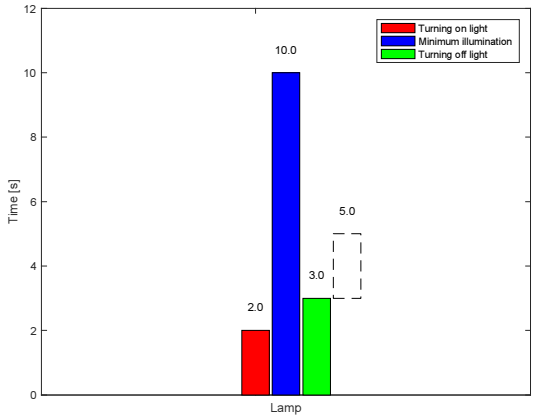

Fig. 9. A technological cycle of a single lamp's operation.

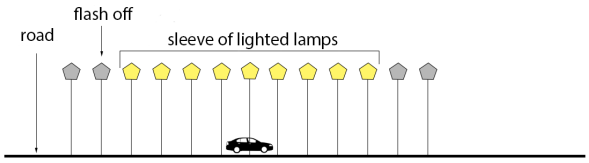

Fig. 10. A diagram of lighting "sleeve" diagram.

3.3. Simulation results

The results of the first simulated experiment with the lighting *on-off wave* (OfW) are shown in Fig. 11, where we can observe the effect of the implemented concept of a lighting "sleeve" (or tunnel), depicted in Fig. 10, which is:

$$R_S = \alpha V_S , \tag{1}$$

were: R_S – a length of the lighting sleeve (the number of lamps being simultaneously on); α – a quantity factor of 0.4318; V_S – a velocity of a moving car.

Within the first 396 metres each car is driving past 9 lamps, located every 44 metres (the lamps are turning on and off in pairs).

The obtained simulation results enable to state that after the car has driven 836 metres, the lamps, starting from the beginning of the road, reach the stable switch-off state. This means that a single car drives in the lighting sleeve of 836 metres (with 19 road lamps on). Four lamps are always on before the approaching car, while the rest (15 lamps) are on at the back of the car.

A length of the lighting sleeve, defined according to the relation (1) as the number of lamps being on at the same time, is dependent on a velocity of a driving car. The value of factor α is determined on the basis of data obtained during simulations, and it represents the ratio of 19 lamps being simultaneously on and the velocity of a driving car. This effective linear relation represents the rule that the higher the car's speed, the longer the lighting sleeve. The executive system coordinator (the central node) switches the lamps on and off in a proper mode, before and after a driving car.

The second experiment concerning the *on-wave* (OnW) has yielded the results, presented in Fig. 12, which show that gaps between the cars are short enough to not switch off the lamps. The lamps are on constantly, waiting for information from the sensors (related to the traffic volume reduction).

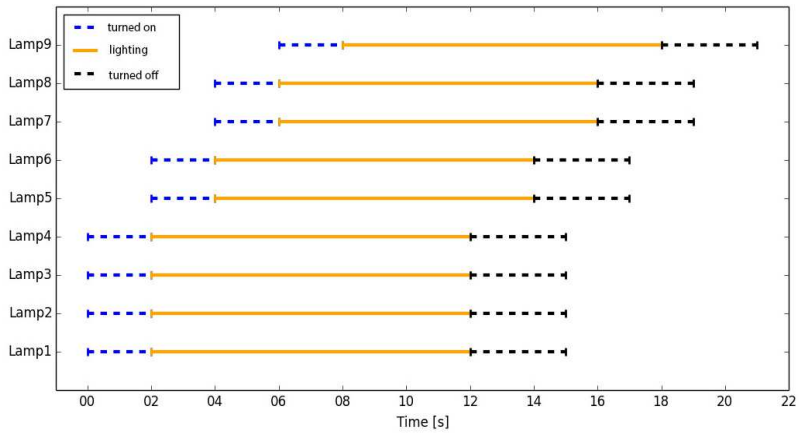

Fig. 11. An off-wave diagram for 9 lamps located along the 396 m distance of the road.

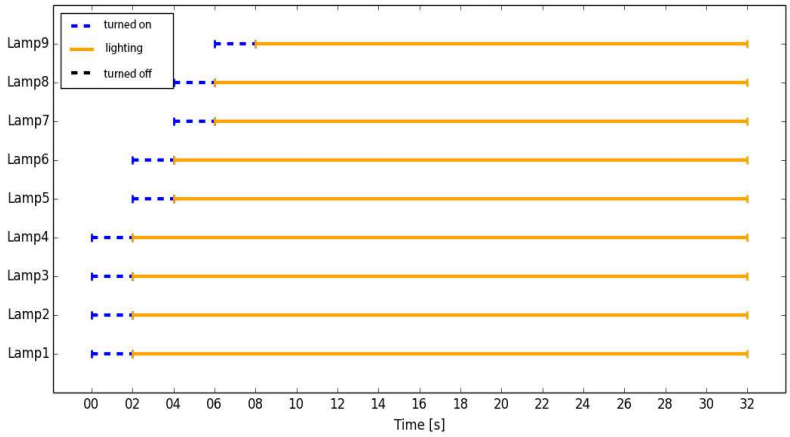

Fig. 12. An On-wave diagram of the lamps located along the 396 m distance of the road.

3.4. Aggregate observations

The simulated examination also covered an aggregated experiment concerning a distance of 10 000 metres (227 lamps). A car driving at a speed of 44 m/s covers this distance in 3 min and 47 sec. Assuming that the car is driving in a 19-second tunnel (836 m or 19 road lamps), the car uses only 8.37% of all available lamps. Obviously, for distances longer than 10 kilometres, the energy gain will be more considerable.

Figure 13 shows a simple impact of the distance on savings expressed in the percentage of lamps being simultaneously on. This impact is now considerable. Naturally, the biggest progress is reached for distances of up to 100 km.

On the basis of common experience, the following optimality criterion can be applied:

$$J = \overline{N}_{on} \cdot P_L \ , \tag{2}$$

were: J – a cost; $\overline{N}_{on}$ – a mean number of turned-on lamps; P_L – a mean power consumption of a single lamp (175 W). This criterion represents a simple function of cost, which is an average power along the highway for a given scenario. It is obvious that for a given lamp technology this criterion can be relatively represented by the average number of lamps turned on.

The criteria have been verified by driving a car at various speeds along the distance of 10 000 m, and the respective analytical data are listed in Table 1.

The results included in Table 1 are presented graphically in Fig. 14, which shows that the higher the speed of a driving car, the higher the number of simultaneously turned-on lamps.

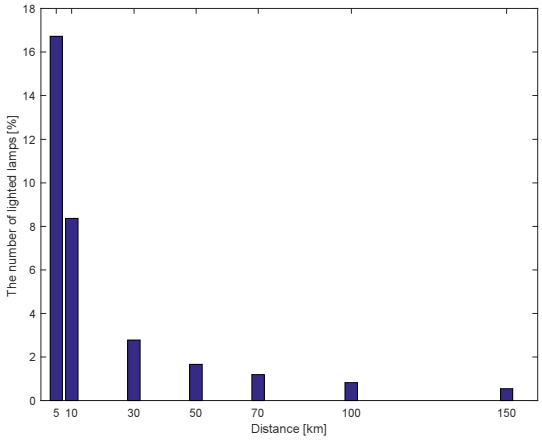

Fig. 13. A percentage of the number of lamps turned on in relation to the total length of the road.

Table 1. Data of the experiment for various speeds of cars along a distance of 10 000 m.

NO.	SPEED	SPEED	TIME OF TRAVEL	NUMBER OF LAMPS TURNED-ON	AVERAGE POWER (W)
1	18 km/h	5 m/s	33.3333 min	6	1 050
2	36 km/h	10 m/s	16.6667 min	7	1 225
3	54 km/h	15 m/s	11.1111 min	9	1 575
4	72 km/h	20 m/s	8.3333 min	11	1 925
5	90 km/h	25 m/s	6.6667 min	12	2 100
6	99.72 km/h	27.7 m/s	6.0168 min	13	2 275
7	104.4 km/h	29 m/s	5.7471 min	14	2 450
8	115.2 km/h	32 m/s	5.2083 min	15	2 625
9	126 km/h	35 m/s	4.7619 min	16	2 800
10	144 km/h	40 m/s	4.1667 min	18	3 150
11	158.4 km/h	44 m/s	3.7879 min	19	3 325
12	180 km/h	50 m/s	3.3333 min	21	3 675
13	198 km/h	55 m/s	3.0303 min	23	4 025
14	216 km/h	60 m/s	2.7778 min	24	4 200
15	234 km/h	65 m/s	2.5641 min	26	4 550

4. Summary

An advantage of the developed research modelling and simulation approach, based on both the SMOL language and the implemented simulation SMOL environment, is the possibility of a convenient description, visualisation and simulation of the analysed *measurement-diagnostics-control* (MDC) networks. It can be used for solving various problems connected with distributed computer systems. The designer of such networks receives a functional tool for an efficient description of the system's, or monitored object's, structures and associated parameters, limitations and relations, as well as for relevant simulations.

The SMOL platform has been developed in order to solve the design problems specific for the distributed network systems. In this paper we have presented an example of analysis of a specific adaptive highway system that proves the multi-purpose character of the SMOL platform. In particular, this platform has been used to analyse the profitability of introducing changes to a road lighting system, which – according to the demonstrated case – confirms the

rationality of building suitable management systems for the lighting of highways used for long and fast driving.

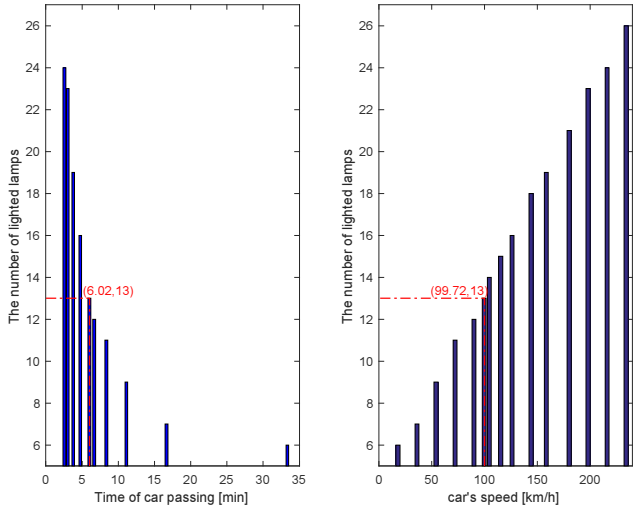

Fig. 14. A mean number of turned-on lamps in the lighting sleeve vs the period of car movement and the car's velocity.

References

[1] Chapman, T. (2008). *Network design language*. *http://www.johntchapman.com/NDL-1.0-WP-080520.pdf*.

[2] DAMADICS. (2004). Research training network on development and application of methods for actuator diagnosis in industrial control systems. *http://diag.mchtr.pw.edu.pl/damadics*.

[3] Dearle, F. (2010). *Groovy for Domain-Specific Languages*. Packt Publishing Ltd., Birmingham, UK.

[4] Doebelin, E. (2003). *Measurement Systems*. (5th ed.). McGraw-Hill Science.

[5] Eckel, B. (2006). *Thinking in Java*. (3rd ed.). Pearson Education, Inc, Upper Saddle River.

[6] Haasz, V. (2012). *Advanced Distributed Measuring Systems*. River Publishers.

[7] Krzaczek, M., Kowalczuk, Z. (2011). Thermal Barrier as a technique of indirect heating and cooling for residential buildings. *Energy and Buildings*. *www.elsevier.com/locate/enbuild*, 43(4), 823–837.

[8] Krzaczek, M., Kowalczuk, Z. (2012). Gain Scheduling control applied to Thermal Barrier in systems of indirect passive heating and cooling of buildings. *Control Engineering Practice*. *http://dx.doi.org/10.1016/ j.conengprac.2012.07.007*, 20(12), 1325–1336.

[9] Kowalczuk, Z., Wszołek, J. (2009). Monitoring objects over networks. *Systems Science*, 35(3), 49–53.

[10] Kowalczuk, Z., Wszołek, J. (2012). Networked Object Monitor – A distributed system for monitoring, diagnostics and control of complex industrial facilities. *Metrol. Meas. Syst*., 19(3), 521–530.

[11] Kowalczuk, Z., Wszołek, J. (2009). Network monitoring and diagnostics of buildings. *Detecting, Analysing and Fault-Tolerant Systems*, Z. Kowalczuk (ed.), PWNT, Gdańsk, 227–234.

[12] Kowalczuk, Z., Wszołek, J. (2014). Modeling of measurement and diagnostics using SMOL language. *Aktualne Problemy Automatyki i Robotyki*, AOW, Warszawa, 277–286.

[13] Nowacki, W. (2006). *Komputerowe Systemy Pomiarowe*. (wyd. II). Wydawnictwa Komunikacji i Łączności, Warszawa.

[14] Łatuszyńska, M. (2011). Computer Simulation Methods – an attempt of logical classification. *Studies & Proceedings of Polish Association for Knowledge Management*, Uniwersytet Szczeciński, Szczecin, 41.

[15] Ważna, W. (2015). Design of an intelligent office. Gdansk Univ. of Technology. Report No. 232 (BSc thesis; under prof. Z. Kowalczuk).

[16] *http://www.cs.put.poznan.pl/mnowak/IB/IB-2.pdf* (Sep. 2016).

Appendix 1: The software code in SMOL language describing the structure of an MDC network (KNX-EIB)

```
package UTest
import SMOH.*
import Utils.*
import com.SMOF.*
import com.UI.DrawSmolEngine;
import com.prototype.DrawSecond;
import com.sample.DrawFirst;
import com.utils.CustomEdge;
import com.utils.INode;
import com.utils.NodeBase;

DrawFirst df = new DrawFirst ()
HashMap<HBase> varList = new HashMap();

// central node definition
def(root) = [new MainNode("ROOT")]
varList.put("root",root);

// KNX/EIB bus definition
def (eib1) = [new EIBNode("KNX/EIB")]
eib1.connect(root ,1,SpeedUnit.Kb,3)
varList.put("KNX/EIB",eib1)

// temperature sensor
// room 1
def (sensor_temp_1)  = [new SANode("SAN_T_P1")]
sensor_temp_1.connect(eib1,1,SpeedUnit.Kb,12)
varList.put("SAN_T_P1",sensor_temp_1)

// room 2
def (sensor_temp_2)  = [new SANode("SAN_T_P2")]
sensor_temp_2.connect(eib1,1,SpeedUnit.Kb,12)
varList.put("SAN_T_P2",sensor_temp_2)

// room 3
def (sensor_temp_3)  = [new SANode("SAN_T_P3")]
sensor_temp_3.connect(eib1,1,SpeedUnit.Kb,15)
varList.put("SAN_T_P3",sensor_temp_3)

// temperature sensor in the air duct
// outdoor temperature
def (sensor_temp_4)  = [new SANode("SAN_K_P1")]
sensor_temp_4.connect(eib1,1,SpeedUnit.Kb,4)
varList.put("SAN_K_P1",sensor_temp_4)

// air temp - IN
def (sensor_temp_5)  = [new SANode("SAN_K_P2")]
sensor_temp_5.connect(eib1,1,SpeedUnit.Kb,3)
varList.put("SAN_K_P2",sensor_temp_5)

// air temp - OUT
def (sensor_temp_6)  = [new SANode("SAN_K_P3")]
sensor_temp_6.connect(eib1,1,SpeedUnit.Kb,3)
varList.put("SAN_K_P3",sensor_temp_6)

// presence detector and light sensor
// room 1
def (sensor_presence_1)  = [new SANode("SAN_OB1")]
sensor_presence_1.connect(eib1,1,SpeedUnit.Kb,6)
varList.put("SAN_OB1",sensor_presence_1)

// room 2
def (sensor_presence_2)  = [new SANode("SAN_OB2")]
sensor_presence_2.connect(eib1,1,SpeedUnit.Kb,13)
varList.put("SAN_OB2",sensor_presence_2)

// room 3
def (sensor_presence_3)  = [new SANode("SAN_OB3")]
sensor_presence_3.connect(eib1,1,SpeedUnit.Kb,13)
varList.put("SAN_OB3",sensor_presence_3)

// motion sensor
// room 1
def (sensor_presence_1)  = [new SANode("SAN_OB1")]
sensor_presence_1.connect(eib1,1,SpeedUnit.Kb,6)
varList.put("SAN_OB1",sensor_presence_1)

// room 2
def (sensor_presence_2)  = [new SANode("SAN_OB2")]
sensor_presence_2.connect(eib1,1,SpeedUnit.Kb,7)
varList.put("SAN_OB2",sensor_presence_2)

// room 3
def (sensor_presence_3)  = [new SANode("SAN_OB3")]
sensor_presence_3.connect(eib1,1,SpeedUnit.Kb,7)
varList.put("SAN_OB3",sensor_presence_3)

// smoke sensor
// room 1
def (sensor_smoke_1)  = [new SANode("SAN_D1")]
sensor_smoke_1.connect(eib1,1,SpeedUnit.Kb,6)
varList.put("SAN_D1",sensor_smoke_1)

// room 2
def (sensor_smoke_2)  = [new SANode("SAN_D2")]
sensor_smoke_2.connect(eib1,1,SpeedUnit.Kb,13)
varList.put("SAN_D1",sensor_smoke_2)

// room 3
def (sensor_smoke_3)  = [new SANode("SAN_D3")]
sensor_smoke_3.connect(eib1,1,SpeedUnit.Kb,13)
varList.put("SAN_D1",sensor_smoke_3)

List<NodeBase> lstBase = Helper.GenerateVertexList(varList);
List<CustomEdge> lstLink = Helper.GenerateEdgeList(lstBase,varList);
DrawSmolEngine dse = new DrawSmolEngine(DrawSecond.DrawDiagram(lstBase,lstLink));
dse.DrawUIGraph();
```

Appendix 2: The software code written in SMOL describing the structure of a highway lighting model

```
package UTest
import SMOH.*
import Utils.*
import com.SMOF.*
import com.UI.DrawSmolEngine;
import com.prototype.DrawSecond;
import com.sample.DrawFirst
import com.utils.CustomEdge
import com.utils.INode
import com.utils.NodeBase

DrawFirst df = new DrawFirst();
HashMap<HBase> varList = new HashMap();

//define central node
def(root) = [new MainNode("ROOT")]
varList.put("root",root);

// LAMP 1
//define profibus node
def (can1) = [new CanNode("LAMP_1")]
can1.connect(root,11,SpeedUnit.Mb,300)
varList.put("can1",can1);

//define SA node
def (sa1) = [new SANode("CD_L1")]
sa1.connect(can1,5,SpeedUnit.Mb,1)
varList.put("sa1",sa1);

//define SA node
def (sa2) = [new SANode("SW_L1")]
sa2.connect(can1,5,SpeedUnit.Mb,1)
varList.put("sa2",sa2);
// End of LAMP 1

// LAMP 2
//define profibus node
def (can2) = [new CanNode("LAMP_2")]
can2.connect(root,11,SpeedUnit.Mb,400)
varList.put("can2",can2);

//define SA node
```

```
def (sa3) = [new SANode("CD_L2")]
sa3.connect(can2,5,SpeedUnit.Mb,1)
varList.put("sa3",sa3);

//define SA node
def (sa4) = [new SANode("SW_L2")]
sa4.connect(can2,5,SpeedUnit.Mb,1)
varList.put("sa4",sa4);
// End of LAMP 2

// LAMP 3
//define profibus node
def (can3) = [new CanNode("LAMP_3")]
can3.connect(root,11,SpeedUnit.Mb,500)
varList.put("can3",can3);

//define SA node
def (sa5) = [new SANode("CD_L3")]
sa5.connect(can3,5,SpeedUnit.Mb,1)
varList.put("sa5",sa5);

//define SA node
def (sa6) = [new SANode("SW_L3")]
sa6.connect(can3,5,SpeedUnit.Mb,1)
varList.put("sa6",sa6);
// End of LAMP 3

// LAMP 4
//define profibus node
def (can4) = [new CanNode("LAMP_4")]
can4.connect(root,11,SpeedUnit.Mb,600)
varList.put("can4",can4);

//define SA node
def (sa7) = [new SANode("CD_L4")]
sa7.connect(can4,5,SpeedUnit.Mb,1)
varList.put("sa7",sa7);

//define SA node
def (sa8) = [new SANode("SW_L4")]
sa8.connect(can4,5,SpeedUnit.Mb,1)
varList.put("sa8",sa8);
// End of LAMP 4

// LAMP 5
//define profibus node
def (can5) = [new CanNode("LAMP_5")]
can5.connect(root,11,SpeedUnit.Mb,500)
varList.put("can5",can5);

//define SA node
def (sa9) = [new SANode("CD_L5")]
sa9.connect(can5,5,SpeedUnit.Mb,1)
varList.put("sa9",sa9);

//define SA node
def (sa10) = [new SANode("SW_L5")]
sa10.connect(can5,5,SpeedUnit.Mb,1)
varList.put("sa10",sa10);
// End of LAMP 5

// LAMP 6
//define profibus node
def (can6) = [new CanNode("LAMP_6")]
can6.connect(root,11,SpeedUnit.Mb,400)
varList.put("can6",can6);

//define SA node
def (sa11) = [new SANode("CD_L6")]
sa11.connect(can6,5,SpeedUnit.Mb,1)
varList.put("sa11",sa11);

//define SA node
def (sa12) = [new SANode("SW_L6")]
sa12.connect(can6,5,SpeedUnit.Mb,1)
varList.put("sa12",sa12);
// End of LAMP 6

// LAMP 7
//define profibus node
def (can7) = [new CanNode("LAMP_7")]
can7.connect(root,11,SpeedUnit.Mb,300)
varList.put("can7",can7);

//define SA node
def (sa13) = [new SANode("CD_L7")]
sa13.connect(can7,5,SpeedUnit.Mb,1)
varList.put("sa13",sa13);

//define SA node
def (sa14) = [new SANode("SW_L7")]
sa14.connect(can7,5,SpeedUnit.Mb,1)
varList.put("sa14",sa14);
// End of LAMP 7

// LAMP 8
//define profibus node
def (can8) = [new CanNode("LAMP_8")]
can8.connect(root,11,SpeedUnit.Mb,200)
varList.put("can8",can8);
//
//define SA node
def (sa15) = [new SANode("CD_L8")]
sa15.connect(can8,5,SpeedUnit.Mb,1)
varList.put("sa15",sa15);
//
////define SA node
def (sa16) = [new SANode("SW_L8")]
sa16.connect(can8,5,SpeedUnit.Mb,1)
varList.put("sa16",sa16);
// End of LAMP 8

// LAMP 9
//define profibus node
def (can9) = [new CanNode("LAMP_9")]
can9.connect(root,11,SpeedUnit.Mb,200)
varList.put("can9",can9);

////define SA node
def (sa17) = [new SANode("CD_L9")]
sa17.connect(can9,5,SpeedUnit.Mb,1)
varList.put("sa17",sa17);
//
////define SA node
def (sa18) = [new SANode("SW_L9")]
sa18.connect(can9,5,SpeedUnit.Mb,1)
varList.put("sa18",sa18);
// End of LAMP 9

List<NodeBase> lstBase = Helper.GenerateVertexList(varList)
List<CustomEdge> lstLink = Helper.GenerateEdgeList(lstBase,
varList)
DrawSmolEngine dse = new
DrawSmolEngine(DrawSecond.DrawDiagram(lstBase, lstLink));
dse.DrawUIGraph();
```

Appendix 3: A Java code generated by the SMOLsim module

```
public class Lamp extends SimulationProcess {
        private LampProcess myModel;
        public boolean turningOnLight = false;
        public boolean minIllumination = false;
        public boolean turningOffLight = false;
        public Lamp(Model owner, String name, boolean
showInTrace) {super(owner, name, showInTrace);
            myModel = (LampProcess) owner;}
        @Override
        public void lifeCycle() {
                sendTraceNote("Lamp process started");
        }
}
```