IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 October 2022, accepted 12 November 2022, date of publication 17 November 2022, date of current version 22 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3222828

RESEARCH ARTICLE

Modeling and Control of Discrete Event and Hybrid Systems Using Petri Nets and OPC Unified Architecture

ERIK KUČERA, OTO HAFFNER, PETER DRAHOŠ, AND ALENA KOZÁKOVÁ
Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, 812 19 Bratislava, Slovakia

Corresponding author: Erik Kučera (erik.kucera@stuba.sk)

This work was supported in part by the Slovak Research and Development Agency under Contract APVV-21-0125; in part by the Cultural and Educational Grant Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic, under Grant KEGA 016STU-4/2020 and Grant 039STU-4/2021; and in part by the Scientific Grant Agency of the Ministry of Education, Research and Sport of the Slovak Republic, under Grant 1/0107/22.

ABSTRACT Discrete event system is a type of system, which changes its state based on asynchronously occuring events. One of many approaches of describing this category of systems is a mathematical formalism called the Petri Net. This article introduces an original software solution, leveraging the graphical representation of this formalism in conjunction with the communication standard OPC Unified Architecture. This combination enables the user to model and control discrete event and hybrid systems using an intuitive, user-friendly graphical interface. The software application also brings new possibilities into scope of this academic domain due to the implementation of a Petri Net formalism extension - continuous elements, which greatly expands the area of systems, which can be modelled and controlled using this tool. The developed software tool was successfully verified in control of a virtual systems. Offering a graphical environment for the design of discrete event / hybrid system control algorithms, it can be used for education, research and practice in cyber-physical systems (Industry 4.0).

INDEX TERMS Discrete-event system, OPC unified architecture, cyber-physical system, system control, hybrid system, Petri nets.

I. INTRODUCTION

The expansion of technology into more and more areas of everyday life brings the need to control these technologies. A term often mentioned nowadays is the fourth industrial revolution - Industry 4.0 [1]. This term covers many different categories of processes, and as each process and technology brings with it certain specificities, the diversity of control methods is also necessary to take these specificities into account. This article is concerned with the design, implementation and subsequent demonstration of a new control method based on a mathematical formalism used to describe processes - Petri net - originally used in the field of discrete event systems, but applicable in an extended form to the modelling of continuous systems.

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaojie Su.

An important type of Petri nets that can be used for systems control are interpreted Petri nets (IPN) [2]. Alternatively, they are also called Petri nets interpreted for control. As the name implies, their variations can be found in the description of software [3], hardware [4] and logic controllers [5], [6]. Their common application is supervisory control [7].

When discrete and continuous parts of a process are combined in a single model, such a model is called a hybrid Petri net [8], [9]. For communication with the controlled system, this work uses the OPC Unified Architecture protocol, which is making its way to the forefront with the ambition of becoming the de facto standard in industrial automation and potentially the standard in IoT and other Industry 4.0 areas [10].

It was crucial to look for projects and literature on the modelling and control of discrete event or hybrid systems using high-level Petri nets during the research process. Finding out whether the current research projects were only interested in

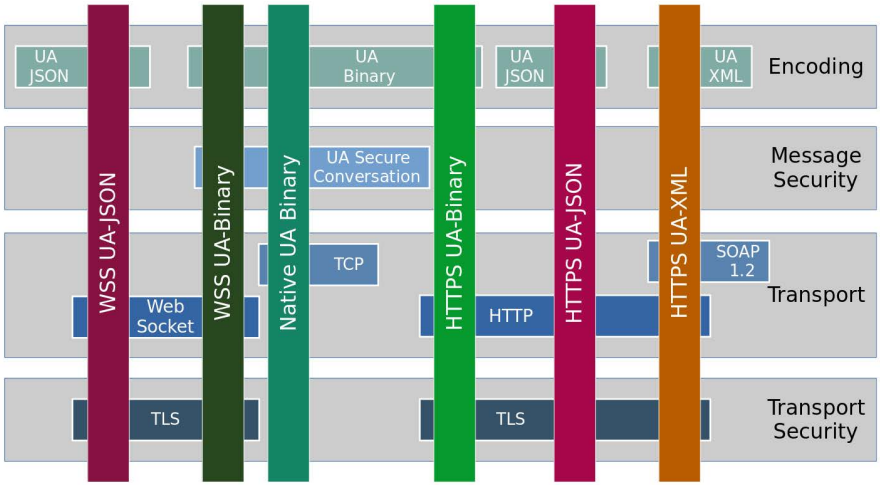


FIGURE 1. OPC UA encoding types [33].

theory or whether some open-source software tools were used or created specifically to support modelling and control using high-level Petri nets was an important question. For the practical outcomes of the presented research project, it would be helpful to find the aforementioned type of research projects.

At first sight, HYPENS [11] appears to be an interesting tool for modelling discrete-time, continuous and hybrid Petri nets. It is an open-source tool for use in the Matlab environment. However, this tool can no longer be downloaded, another disadvantage of this software was that in its environment the modelling of Petri nets was only possible using their matrix representation, not taking advantage of one of the great properties of Petri nets - the possibility of an easy to understand graphical representation of the model. The possibility of running this tool in current versions of Matlab is also unclear. Other tools for modelling using hybrid Petri nets include, for example, SimHPN [12]. Advantages of this tool are various analytical tools for working with Petri nets and cooperation with Matlab. The disadvantage is that one of the pillars of Petri nets is lost, namely the graphical representation. Petri nets are only displayed in the form of matrices. It is also not clear whether compatibility with current versions of Matlab is ensured.

The use of the concept of Petri nets in conjunction with control is still a largely unexplored area that has not yet received much attention from the lay or academic community. There are works that address this area from several perspectives, but these often do not include the software described in these works, either in executable form or in source code form. In the field of cybernetics, in papers [13], [14] the authors describe the use of hybrid and coloured Petri nets for the purpose of traffic modelling on intersections and highways. In other work, the same authors address interesting topics in the field of manufacturing plants [15], [16]. It is not possible to determine from these works whether it is only a theoretical exploration of the field, or whether their findings have also been used in practical applications.

In [17], a tool is described whose basic idea is to control the system from a computer, through an Arduino microcontroller, using the Firmata protocol. Management of control systems using microcontrollers is not a suitable approach for industrial use, since industrial applications generally use PLC-type computers. Moreover, the approach described in this work requires specific firmware modification in the microcontroller, which may not be an easy task for the purpose of wider deployment on a large amount of hardware.

In [18], the authors developed the Matlab application RCPetri, enabling graphical modelling, simulation and synthesis of control algorithms for PLC-type computers. This software also includes a module allowing communication with the controlled system via Modbus and OPC UA protocols. However, this work has the disadvantages of an opaque user interface and the necessity of a textual definition of the process through a specific Microsoft Excel document. Additionally, from the content of this work, it appears that this application does not support Petri net's arc weighting, and has support for marking with only 0 and 1 values.

This summary (Table 1) clearly illustrates the lack of availability of software tools with interfaces allowing direct control of systems. The tool proposed in this work has advantages over RCPetri, particularly in supporting continuous and hybrid Petri nets and extended arc types. Another difference is the presence of an easy-to-use, integrated user interface, eliminating the need to create specific external documents. The developed user interface makes use of the capabilities of OPC UA protocol - for example, for the purpose of retrieving available variables corresponding to, for example, sensors and actuators connected to the PLC. OPC UA communication is assignable to places and transitions of Petri net. In contrast, our work does not aim at synthesising code directly executable from the PLC environment, which RCPetri tool enables.

Since the developed application described in our article should be compatible with industrial automation,

TABLE 1. Summary of selected Petri net editors.

TOOL NAME	ADVANTAGES	DISADVANTAGES
HYPENS [11]	Modelling of discrete, continuous and also hybrid Petri nets, open-source, cooperation with Matlab	Petri net model can be defined only using matrices, no graphical representation, questionable support of new Matlab versions
SimHPN [12]	Modelling of discrete, continuous and also hybrid Petri nets, analytic tools, cooperation with Matlab	Petri net model can be defined only using matrices, no graphical representation, questionable support of new Matlab versions
RCPetri [18]	Graphical modelling, simulation and synthesis of control algorithms for PLC-type computers, module allowing communication with the controlled system via Modbus and OPC UA protocols	Opaque user interface and the necessity of a textual definition of the process through a specific Microsoft Excel document, does not support Petri net's arc weighting
Snoopy [19]	Modelling of wider range of Petri nets - stochastic, hybrid, colour, music, etc.	Source code is not available, mainly for biology applications (not for system control)
Visual Object Net++ [20]	Modelling of discrete, continuous and also hybrid Petri nets, many scientific papers that deals with this tool with examples	Not open-source and not further developed
GPenSIM [21]	Modelling of discrete, timed Petri nets, many options for simulation and analysis	Petri net model can be defined only using matrices, no graphical representation, model and software cannot be used for system control
CPN Tools [22]	Modelling of different types of coloured Petri nets, good analytic tools	Model and software cannot be used for system control
PIPE2 [34]	Modelling of discrete, many options for simulation and analysis, open-source and still developed	No options for system control
Renew [36]	Modelling of different types of coloured Petri nets, open-source and still developed	No options for system control

it is necessary to indicate which communication protocols are used in this area. Despite the large number of available protocols, the majority of wired protocols follow either the Fieldbus or Industrial Ethernet standards. For our application, the relevant protocols are from the second group.

As the first, we can mention PROFINET [23]. In general, PROFINET networks are quicker than those of its rivals. Siemens provides a selection of PROFINET-capable hardware that makes it simple to upgrade or replace Siemens control systems. However, when other brands are involved, upgrades are less adaptable. However, PROFINET infrastructure is generally more expensive than other types. Although tree and star topologies are also supported, line topology is typically used to deploy PROFINET networks because it requires less cable.

Due to the use of commercially available components and its object-oriented foundation, Ethernet/IP [24] is the more adaptable and compatible protocol. It is slower than PROFINET but also more economical. Although ring and tree topologies are frequently used in Ethernet/IP networks to increase redundancy, star topologies are their preferred configuration.

An important protocol is Modbus TCP/IP [25]. This protocol is an improved version of Modbus RTU with a TCP interface. Modbus-formatted data is processed, but it is wrapped and sent over Ethernet. In contrast to PROFINET and Ethernet/IP, which are merely configured using a configuration tool, a Modbus TCP network is typically programmed directly into the controller's source code using function blocks or other libraries.

An interesting protocol is EtherCAT [26]. This protocol substitutes Ethernet communication for TCP/IP, allowing a telegram to build on itself as it travels through the network after each node. The telegram returns to the EtherCAT master at the end of the cycle. Due to the fact that a message only needs to pass through each node once, communication is very effective. The determinism levels that EtherCAT nodes can adhere to are in the range of a few milliseconds. However, because special EtherCAT-ready devices must be used, EtherCAT is not a flexible network. Despite the higher average cost of EtherCAT devices, there are some cost savings in a network as a whole because there are fewer components required. Managed network switches, for instance, are not required.

In the presented application OPC UA communication standard was used, which is described in the next section.

II. MATERIALS AND METHODS

The aim of this work is to design and implement a solution that allows the use of the mathematical formalism of discrete, continuous, or hybrid Petri nets in system control.

A. OPC UNIFIED ARCHITECTURE

The specific focus of this work is on the control of industrial automation systems through the control of PLCs from a standard computer environment. This will be done without the need for PLC software programming, which allows the tuning of control algorithms through changes in the control Petri net, a much quicker response to potential problems, and also makes it possible to use processing and memory resources of a standard computer. The connection with the graphical interface brings high user friendliness. Since modern PLCs include a server supporting OPC UA protocol [27], we decided to use it. The disadvantage of this approach may be the impossibility of real-time control, if the surrounding

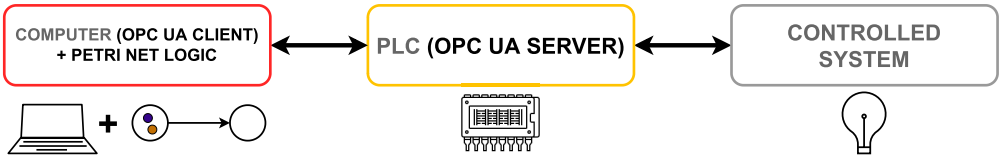


FIGURE 2. Basic scheme of proposed solution.

TABLE 2. Summary of selected industrial communication protocols.

PROTOCOL NAME	ADVANTAGES	DISADVANTAGES
PROFINET [23]	Faster than competitors, big market share, support of big company (Siemens)	More costly
Ethernet/IP [24]	More flexible and compatible protocol, cheaper than PROFINET	Slower than PROFINET
Modbus TCP/IP [25]	Lightweight and simple, availability of open source libraries	Quite old protocol, weak security
EtherCAT [26]	Unique method of Ethernet communication in which a telegram traversing through the network can build upon itself after each node, very fast protocol, cost savings thanks to reduced number of components needed	Not flexibile because of special EtherCAT-ready devices need to be implemented
OPC Unified Architecture [27]	High-level security, standard for Industry 4.0, vendor interoperability, possibility of OPC UA communication via time-sensitive networks	Some advanced functionalities still in development

infrastructure, including the computer and operating system on which the control algorithm runs, is not adapted for such operation. However, this limitation can be minimised or eliminated by using means supporting real-time communication, such as real-time operating systems, industrial Ethernet and others [28].

The original OPC (OLE for Process Control) standard appeared in 1995 and allowed interoperability and secure data exchange between operating devices from different manufacturers and any client application [29]. Systems supporting the original version of OPC provide a standardized way of accessing data from industrial processes, which prior to the emergence of this standard was hampered by differences between proprietary approaches from different manufacturers.

The current OPC UA (Open Platform Communication Unified Architecture) standard was released in 2006. This standard provides specifications for the exchange of information in industrial communication, and beyond. It is platform-independent, suitable for a variety of device types, from embedded systems to large cloud solutions, which promotes interoperability. Its development is actively supported by the OPC Foundation. Compared to the original OPC, it retains the features of secure and reliable data transfer between operational devices, and introduces enhancements especially in the transfer mechanisms and data modelling [30].

OPC UA defines the possibility of access using client-server architecture, and also the publish-subscribe mechanism. In the case of using the client-server architecture, the server provides services that clients can interact with (centralized architecture, one-to-many or many-to-one access). In contrast, publish-subscribe is more suitable for situations where many-to-many communication is required, i.e. data is distributed from multiple publishers to multiple subscriptions. The standard defines 3 data encoding models - UA Binary, UA XML and UA JSON [31]. Data encoded in this way can then be transmitted using TCP, WebSocket Secure, or HTTPS protocols - Fig. 1.

This standard also includes a prescribed method of data modelling, enabling the process data provided by industrial equipment to be effectively captured in a structured form. For this purpose, the OPC UA standard defines a hierarchical structure consisting of nodes and mutual references between these nodes. The information model made according to this standard is then stored in the address space of the OPC UA server, where it can be accessed by any OPC UA client. OPC UA also defines so-called companion specifications that define the information model and its semantics for specific domain areas, which again facilitates interoperability [32].

B. SELECTION OF EXTENSIBLE PETRI NET EDITOR

Several freely available Petri net editors were analyzed for the basis of this work. Since practically the entire further implementation part depended on this choice, it required careful consideration. The editors analyzed included PIPE2 [34], [35], Renew [36], CPN Tools [37], and PNEditor [38]. The selection conditions were:

- Friendly and simple user interface
- Availability of source codes
- Multiplatform
- Cleanliness of the source code and its extensibility

We wanted to realize the solution in such a way that it was as easy as possible to implement, extend and use for educational purposes. That is why we wanted to use the Petri net editor, whose code is written in a clear and understandable way, as a base. This criterion was more important to us than any advanced analytical functionality of the editor.

After considering these criteria, we decided to use the PNEditor editor as the basis of our implementation. PNEditor is a graphical Petri net editor programmed in Java, which makes it multiplatform. The clean design of this software, the adherence to design patterns and the clarity of the code are huge advantages. In particular, because of the use of the *Action* and *Command* design patterns [39], it is very easy to extend the actions that this editor allows, while removing functionality that is not necessary or appropriate for the purposes of this work.

Basic scheme of proposed solution can be seen in Fig. 2.

III. IMPLEMENTATION

At the beginning of the technical part of the paper it is necessary to mention the main application benefits of the developed solution:

- The possibility of implementing a control algorithm using Petri nets and the advantages that this mathematical formalism brings with it.
- The use of hybrid Petri nets, which brings the possibility of control not only discrete event systems, but also systems with continuous components.
- The use of Petri nets and their graphical representation allows a simple and quick change of the control algorithm using a graphical editor.
- The developed solution communicates using the universal and widely adopted OPC Unified Architecture communication standard.

This section will describe the implementation and functionality of a new tool for system control using hybrid and discrete Petri nets. Communication takes place via OPC UA.

A. HYBRID PETRI NETS

Since the selected Petri net editor PNEditor did not include support for continuous and hybrid Petri nets [40], this was the first task that needed to be addressed. Initially, the marking and weighting values of all places and arcs were represented using the *int* data type, a Java representation of a primitive data type used for storing integer values. At first glance, the reader may find it easy to change this data type to another primitive data type used for storing decimal numbers, such as the *float* and *double* types in Java, but these are not suitable for use for this purpose because these values are not only stored in the application, but arithmetic operations are also performed on them. Because of the way in which the storage of such variables is implemented at the hardware level, it is impossible to represent some values accurately in mathematical operations on them. For Java (and also other languages running in a virtual machine), we are not referring to a direct hardware implementation, but to an implementation in the virtual machine in which the program is run.

For these reasons, we decided to use the *BigDecimal* class from Java standard library to store these numeric values, whose original purpose was to provide a way to implement mathematical operations that depend on very high precision,

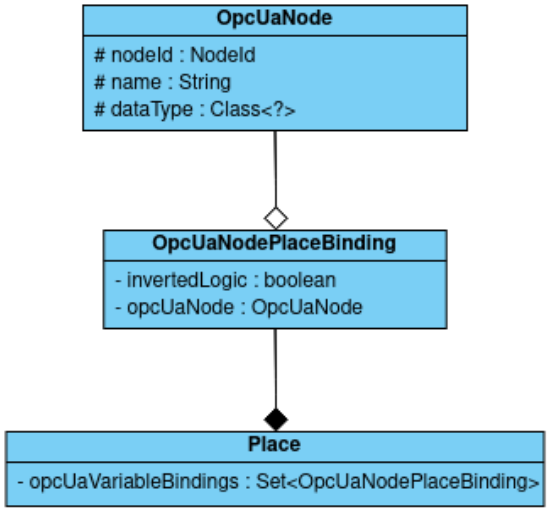


FIGURE 3. Class diagram - assigning OPC UA variable to Petri net place.

such as financial operations. Since objects are handled differently in Java than primitive data types, this modification required identifying all user-accessible entry points in the application, as well as modifying the implementation of all arithmetic operations, from the triggering of Petri net transitions, including the correct behaviour in the case, to the behaviour of extended arc types. This change also required modifying the serialization of the classes representing points and edges for the purpose of storing them in *pflow* format.

B. OPC UA CLIENT

Another of the tasks set out for this work was to enable systems control using Petri net formalism via OPC UA protocol. For this purpose, it was necessary to first define the relationship between the variables located in the address space of the OPC UA server and the elements of the Petri net. A natural way, which at the same time does not contradict the formal properties of Petri nets, turned out to be to ''pair'' the marking of selected places with selected variables on the server, and to write the same marking to a variable on the server whenever the marking (value) of a relevant place changes.

At first glance, it may seem that for this principle to work, it is absolutely necessary that the values of variables in the OPC UA server environment are not changed during the control by other clients. The solution to this problem brings us to the involvement of the role of transitions in the Petri net in the operation of this mechanism, in a way where a transition is triggered by a user-defined event associated with a variable of the OPC UA server. In this way, we can use such external changes (they can be, for example, changes in the values sensed by the sensors) in the control. Since variables in the OPC UA standard are typed, and the editor at this point allows to use real-number marking of places in addition to integer marking, it is necessary to define a mapping between the place marking and the value written to the variable on the server.

TABLE 3. Mapping of OPC UA data types and examples of marking values.

Data type of OPC UA variable	Marking value	Value written to OPC UA variable
Boolean	0 >0	false true
SByte Byte Int16 UInt16 Int32 UInt32 Int64 UInt64	0 0.4 0.5 20.5 70	0 0 1 21 70
Float Double	0 0.4 0.5 20.5 70	0 0.4 0.5 20.5 70

In this paper, we propose and implement the following mapping - Table 3.

For data types defined in OPC UA standard other than those listed in Table 3, our implementation does not define a mapping, and if a variable is of a different type, the user interface does not allow such a variable to be assigned to a place or transition.

We chose the Eclipse Milo library [41] as the basis for the actual implementation of the OPC UA client interface. There are several Java libraries providing similar functionality, but none of them reaches the quality of Eclipse Milo in terms of price, open source, complete documentation, existence of implementation examples, active development and support from its author. In addition, it wins when comparing data throughput with other implementations [42]. Eclipse Milo is a project developed under under the auspices of the Eclipse Foundation, and its terms of use are defined by the license EPL-2.0 [43].

The actual implementation of how OPC UA server variable is assigned to Petri net place is illustrated in Fig. 3.

After each marking change, if such an assignment exists, the current marking value on the relevant Petri net place (after the mapping corresponding to Table 3) is written to a variable on OPC UA server that is uniquely identified by its *NodeId*, which is stored in *OpcUaNode* object.

The implementation of assigning events to transitions is a bit more complex. The basis of the implementation is the abstract class *Operator*, and its concrete implementations (Fig. 4). In this figure, the blue arrows indicate ''extends'' and the green arrows indicate ''implements''. The figure was generated directly in the IntelliJ IDEA development environment [44]. The way, in which the connection to *Transition* class is implemented, is illustrated by the diagram in Fig. 5.

The basic principle that is used in the automatic triggering of transitions is publish-subscribe principle, which is implemented by Eclipse Milo library through *ManagedSubscription* class. Using this class, it is possible to ask the server to send information to our application (as a client) whenever the value of a variable changes. This variable is again specified by its *NodeId*.

Then the application evaluates whether the received value meets the condition defined in the *OpcUaVariableCondition* class. If it does, it attempts to trigger (fire) the transition that is bound to this condition. Of course, such a transition is only triggered if the conditions defined by Petri net formalism are also satisfied, i.e., all the entry places of the transition have enough tokens (marking), depending on the weight of the entry arc of this transition.

Using the graphical user interface (GUI), it is also possible, in addition to the above assignments, to specify minimum sampling interval the server should send potential changes to the values of a variable that is marked as ''of interest'' using the publish-subscribe mechanism. This value can be specified in common for all conditions during the connection to OPC UA server (Fig. 6). It is also possible to specify a separate value for each defined condition, using GUI used for assignment (Fig. 7). This is then stored in the variable *samplingInterval* (in the *OpcUaNodeTransitionBinding* class). This value is taken into account independently of the global sampling interval.

C. MODIFICATIONS OF PNEditor's GRAPHICAL USER INTERFACE

The following section summarizes the graphical modifications in the editor environment that have been made to enable the use of aforementioned functionalities.

1) SETTING PETRI NET ELEMENTS TO CONTINUOUS

To allow changing the place or arc to continuous, this option has been added to the context menu of the individual elements on the canvas (available by right-clicking - Fig. 8). To maintain compatibility with the way the original editor functionality works, the same options are also accessible in the panel at the top of the window, under the *Element* menu item (Fig. 9). These buttons work in a ''toggle'' manner, i.e. switching between discrete and continuous place/arc types, again in line with the existing buttons of the same menu. An arc can only be set as continuous if the place to which it is connected is set as continuous. Also, the GUI will not allow the location from which a continuous arc originates to be set as discrete.

2) GRAPHICAL ELEMENTS RELATED TO OPC UA CONNECTION

Another significant change in the GUI is the addition of an item in the main manu - *OPC-UA* and the buttons associated with this item (Fig. 10). You can use this menu to access the editor functionality related to the connection to OPC UA server. Using the *OPC-UA server connection...* item, it is possible to display a modal dialog box, which is used to configure the address of the OPC UA server. In this window it is also possible to set the global sampling interval valid for all connection transitions and variables of the OPC UA server (Fig. 6).

FIGURE 4. Class diagram - Operator class and its implementations.

Once connected to the server, you can view and assign OPC UA address space variables to locations and transitions of the Petri net by selecting the item *Edit OPC-UA bindings...* in this menu (Fig. 7). If this window is accessed from the top panel, the listbox at the top of the window allows to switch between all places and transitions of the Petri net. This same window can also be accessed by selecting the *Bind OPC-UA variable* option in the context menu of any of the places and transitions on the canvas. In this case, the Petri net element from which this window was invoked is preselected in the top listbox and cannot be changed.

The middle panel of this window changes depending on whether a place or a transition is selected in the top listbox. In the case of places, it is only possible to simply assign a variable to which the marking of the place will be written; in the case of variables with *Boolean* data type, it is possible to additionally select whether the marking value should be written directly (corresponding to Table 3) or inverted, i.e. the opposite value to the markup value, using the *Inverted* checkbox.

In the case of transitions, depending on the data type of the variable, it is possible to define, by selecting an operator-value pair, a condition that should cause the transition to be triggered. In this case, it is also possible to specify an individual sampling interval for each transition triggering (firing) condition, different from the globally defined one (Fig. 7).

The last menu item in the top panel is a checkbox that the user can use to hide or show the rendering of existing site assignments and transitions (Fig. 11).

IV. CASE STUDIES

Virtual models of a discrete event and hybrid system were used to validate the functionality. For this purpose, the Factory I/O simulation software [45] was used as a training software for teaching the programming of control algorithms. It is also widespread in use as training software for positions of workers managing the physical operation of automated factories or assembly shops. It can be used to model common components of automated operation, including sensors or actuators. It can simulate real interactions between these components and also fault conditions. Factory I/O supports control input from a number of real PLC control circuits, but also using Modbus, OPC DA and OPC UA protocols.

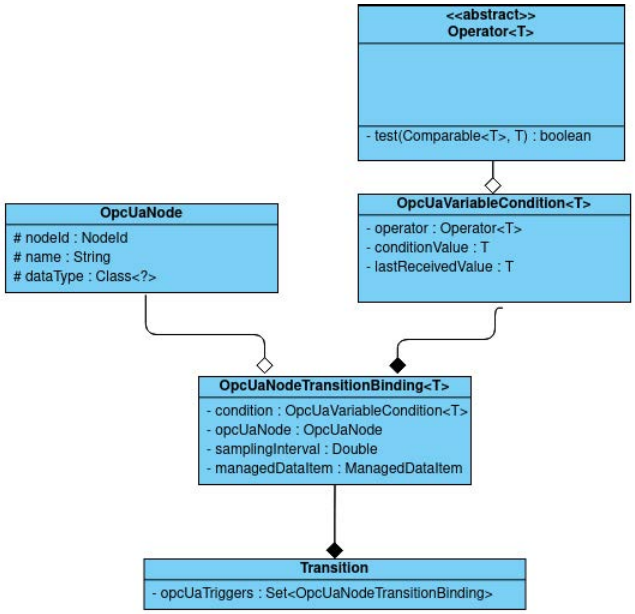


FIGURE 5. Class diagram - assigning event to Petri net transition.

Of course, for us the support of the OPC UA protocol was relevant.

For the demonstration, 2 pre-prepared automated lines (called scenes in the Factory I/O environment) were selected, which will be introduced later in the text.

Our implemented solution (extension of PNEditor) is OPC UA client (since PLC type computers usually provide OPC UA server), and Factory I/O is also implemented as a client. Direct client-to-client communication is not possible using the protocol, so we needed to use an intermediate link (middleware) to simulate the operation of OPC UA server to validate our solution. There are several solutions for this, either from Unified Automation [46], or it is possible to program the server [47]. For the purposes of experimental validation, both of these approaches proved to be complex and time consuming, with no obvious benefit to presented work and article.

By further research, we found a more suitable solution based on the Node-RED software [48]. Node-RED is both a platform and a tool for flow-based programming. It is a programming paradigm that allows to describe the behaviour of an application as a network of nodes. Each of the nodes has a clearly defined role; data enters the node, the node modifies the data, and again the modified data exits the node. It is the responsibility of the network of nodes to appropriately transfer this data between the nodes. This programming paradigm is very easy to visualise and therefore more accessible and understandable to a wider range of users [49] (Fig.12).

The basic functionality of Node-RED does not provide the ability to configure and run an OPC UA server. The *node-red-contrib-opcua* add-on package was used for this purpose. The method of configuring the address space of the OPC UA server included in this package is illustrated here:

```
msg.payload = { "opcuaCommand":
"addVariable" }
msg.topic = "ns=1;s=variableName;
datatype=Boolean"
```

The example illustrates the format in which configuration data is transferred between an *inject* node and an *OpcUa-Server* node.

The architecture of the test environment can be seen in Fig. 13.

After configuring the address space of the OPC UA server using the above syntax, modeling the Petri net and creating the appropriate interconnections, correct automatic control takes place for the following case studies.

A. CASE STUDY: CONTROL OF DISCRETE EVENT SYSTEM USING PETRI NET

The first case study uses a scene from Factory I/O called *Sorting by Height (Basic)*. It is an industrial line used for sorting of packages by their height. This is done based on input from a light sensor. Using this line, the functionality of interfacing discrete Petri nets with OPC UA protocol was demonstrated (Figure 14).

In terms of the structure of the Petri net modelled for the control of *Sorting by Height*, these are 3 disjunctive nets that could also run in separate control processes. However, for clarity we have placed them on a single canvas (Fig. 15). Only discrete places and simple arcs are used in this Petri net, all with weight 1. The places of Petri net are described in Table 4. The transitions of Petri net are described in Table 5.

All places in Petri net (with the exception of the *Sensed low* and *Sensed high* places) correspond to the state of one of the actuators (conveyors) of the system, and thus there is an assignment to a variable of the address space of the OPC UA server, which is assigned on the software side of the Factory I/O to control these actuators.

Transitions in Petri net are in contrast assigned to the sensor members of the system. And when a defined condition is met, they react to a change in the state of the system - they reflect it in the corresponding places of Petri net.

For illustrative purposes, some of these assignments are duplicated in our case study. For example, in Petri net detail in Fig. 16, the variable *fiooEntryConveyor* does not need to be assigned to both the *Entry Conveyor running* place and the *Entry Conveyor stopped* place, with the flag of writing an inverted marking value. In this case, the same value is written to the variable twice, and control of this system would of course be functional without this redundancy.

The *Sensed low* and *Sensed high* places, and the way they are connected in the Petri net, serve a purpose analogous to the Boolean variable used in other forms of control for the purpose of deciding which way the package should proceed, i.e., how the state of the system should evolve.

The control of the *Sorting by Height* scene starts with a situation where the entry conveyor belt is started by automatic detection using the *At right entry/At left entry sensors* and the corresponding assignment to the *Start entry conveyor*.

TABLE 4. Description of Petri net places for first case study - Sorting by Height (Basic) scene.

PLACE ID	PLACE DESCRIPTION	CONNECTION WITH OPC UA VARIABLE
Sensed low	Records the passing package's height for the purpose of further decision where to send it.	-
Sensed high	Records the passing package's height for further decision where to send it.	-
SendLeft stopped	When this place's marking >0, the value of the variable controlling the chain transfer's left direction is set to *false* due to the *inverted* flag, representing a stopped state.	fiooSendLeft (inverted)
SendLeft running	When this place's marking >0, the value of the variable controlling the chain transfer's left direction is set to *true*, representing a *running* state.	fiooSendLeft
On left conveyor	The marking of the place represents the number of packages currently present on the left conveyor and is used for determining the conveyor's desired state. Marking >0 ? left conveyor running : left conveyor stopped	fiooLeftConveyor
SendRight stopped	When this place's marking >0, the value of the variable controlling the chain transfer's left direction is set to false due to the *inverted* flag, representing a *stopped* state.	fiooSendRight (inverted)
SendRight running	When this place's marking >0, the value of the variable controlling the chain transfer's right direction is set to *true*, representing a *running* state.	fiooSendRight
On right conveyor	The marking of the place represents the number of packages currently present on the right conveyor and is used for determining the conveyor's desired state. Marking >0 ? right conveyor running : right conveyor stopped	fiooRightConveyor
Entry Conveyor stopped	When this place's marking >0, the value of the variable controlling the main entry conveyor's state is set to *false* due to the *inverted* flag. Represents the main entry conveyor's *stopped* state.	fiooEntryConveyor (inverted)
Entry Conveyor running	When this place's marking >0, the value of the variable controlling the main entry conveyor's state is set to *true* representing a *running* state.	fiooEntryConveyor
Load conveyor stopped	When this place's marking >0, the value of the variable controlling the state of the conveyor which is responsible for loading the package on top of the chain transfer is set to *false* due to the *inverted* flag. Represents the *stopped* state of the conveyor which is responsible for loading the package on top of the chain transfer.	fiooLoadConveyor (inverted)
Load conveyor running	When this place's marking >0, the value of the variable controlling the state of the conveyor which is responsible for loading the package on top of the chain transfer is set to *true*, representing a *running* state.	fiooLoadConveyor

TABLE 5. Description of Petri net transitions for first case study - Sorting by Height (Basic) scene.

TRANSITION ID	TRANSITION DESCRIPTION	CONNECTION WITH OPC UA VARIABLE
SL	Triggered when the height sensor's lower part is intersected by a package.	fioiLowSensor == true
SH	Triggered when the height sensor's higher part is intersected by a package.	fioiHighSensor == true
Start left cycle	Starts the process of sending a package to the left using chain transfer. Triggered when the package reaches the "Loaded" sensor.	fioiLoaded == true
Stop sendLeft	Triggered when a package reaches the sensor at the start of the left conveyor. Stops the chain transfer.	fioiAtLeftEntry == true
Remove from left conveyor	Triggered when a package reaches the sensor at the end of the left conveyor. Decreases the marking of the place "At left conveyor" and in case the marking of this place is zero, meaning there are no more packages on the left conveyor, stops the left conveyor.	fioiAtLeftExit == true
Start right cycle	Starts the process of sending a package to the right using chain transfer. Triggered when the package reaches the "Loaded" sensor	fioiLoaded == true
Stop sendRight	Triggered when a package reaches the sensor at the start of the right conveyor. Stops the chain transfer.	fioiAtRightEntry == true
Remove from right conveyor	Triggered when a package reaches the sensor at the end of the right conveyor. Decreases the marking of the place "At right conveyor" and in case the marking of this place is zero, meaning there are no more packages on the right conveyor, stops the right conveyor.	fioiAtRightExit == true
Start load conveyor	Starts the conveyor responsible for loading the packages on top of the chain transfer. Triggered together with setting the state of the main entry conveyor.	fiooEntryConveyor == true
Stop load conveyor	Stops the conveyor responsible for loading the packages on top of the chain transfer. Triggered when a package reaches the "Loaded" sensor.	fioiLoaded == true
Start entry conveyor	Starts the main entry conveyor. Triggered when a package reaches one of the sensors at the beginning of either of the side conveyors.	fioiAtLeftEntry == true , fioiAtRightEntry == true
Stop entry conveyor	Stops the main entry conveyor. Triggered when a package reaches the "Loaded" sensor.	fioiLoaded == true

During the passage of the pallet with the package through the light gate, it is recorded whether the package has crossed the lower or also the higher positioned sensor. When the pallet arrives at the chain rectifier, at the end of which there is a sensor called *Loaded* tracking the presence of the pallet (represented in the network as the condition *fiooLoaded*==*true*), the entry conveyor is stopped by automatically firing the *Stop entry conveyor* transition. And the pallet, based on the previous recorded height sensor value, is sent to the right or left conveyor. These conveyors are only switched on if there is a package on them that has not yet reached the end of the conveyor, which is monitored by the *At left exit/At*

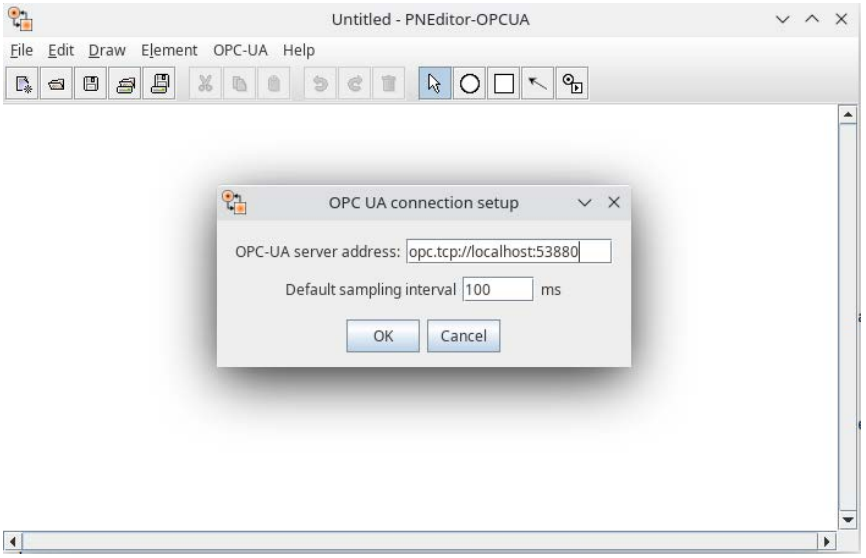


FIGURE 6. Graphical user interface - connection to OPC UA server.

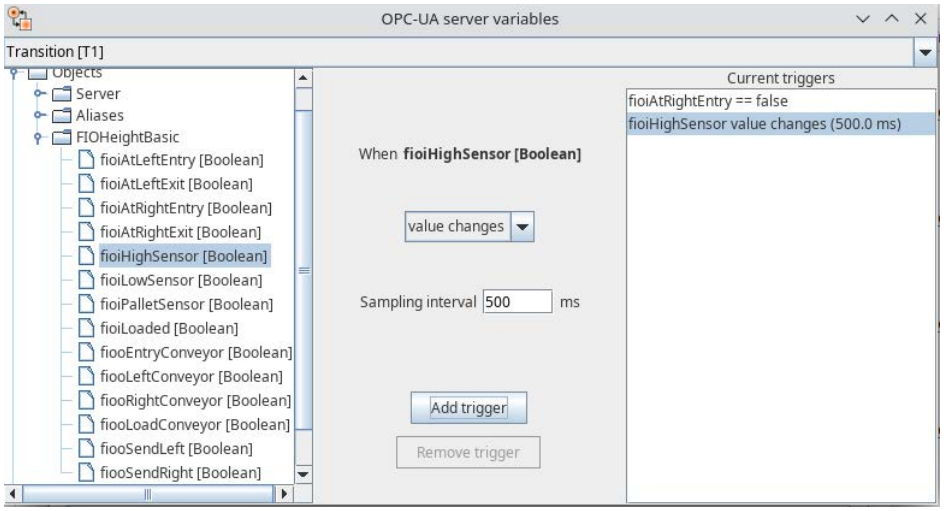


FIGURE 7. Graphical user interface - variable assignment dialog - transition view.

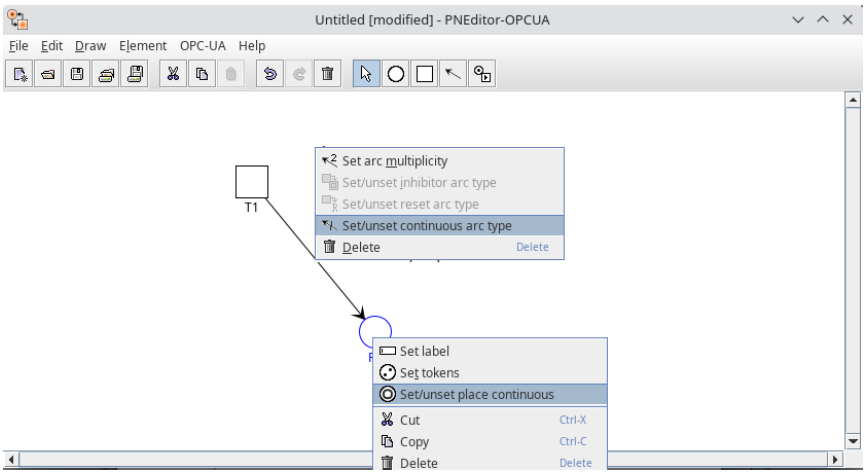


FIGURE 8. Graphical user interface - setting the arc/place of Petri net to a continuous type.

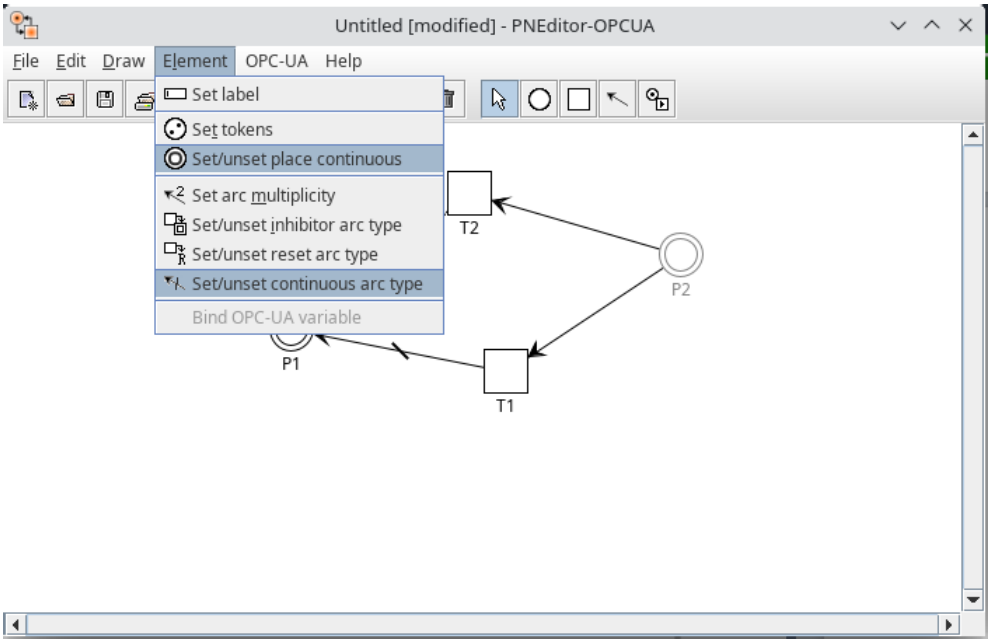


FIGURE 9. Graphical user interface - setting the arc/place of Petri net to a continuous type.

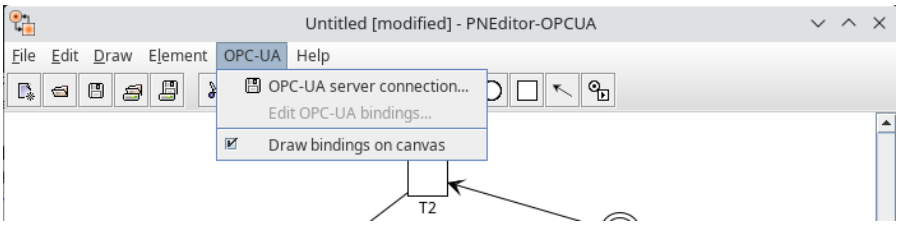


FIGURE 10. Graphical user interface - OPC UA - main menu.

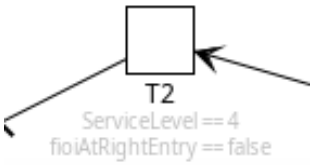


FIGURE 11. Graphical user interface - transition with associated firing conditions.

right exit sensors, and the number of these packages on each conveyor can be monitored at the *On left conveyor* and *On right conveyor* places of Petri net.

If the pallet with the package reaches the end of the conveyor, the marking is removed from the conveyor by automatically firing the *Remove from left conveyor/Remove from right conveyor* transition again. And the value of the marking located at the corresponding place (according to the mapping shown in Table 3) is written to the corresponding variable on the server. This entire decision process is cyclically iterated.

We can conclude that the ability of discrete event control with our extended software application was successfully verified and can be generalised for other applications.

B. CASE STUDY: CONTROL OF HYBRID SYSTEM USING HYBRID PETRI NET

The second case study uses a scene from Factory I/O called *Assembler (Analog)*. It is an industrial line, the purpose of which is to assemble a part from a base and a lid by means of a two-axis arm controlled by analogue values. This means that this case required a functional implementation of continuous or hybrid Petri nets and their interfacing with OPC UA protocol (Fig. 17).

To model the Petri net used to control the *Assembler (Analog)* system (Fig. 18), it was necessary to use a wider range of editor functionalities, including continuous places and arcs, and also reset arcs. During modelling, it was also necessary to determine the values of the continuous arcs weights corresponding to the displacements/increments of the X and Z coordinates of the biaxial arm - this was done experimentally. The places of Petri net are described in Table 6. The transitions of Petri net are described in Table 7.

To start the control, you need to set the initial marking of Petri net manually by triggering (firing) the *Start* transition. This has to be done in order to bring the industrial line, and therefore the corresponding variable values in the OPC

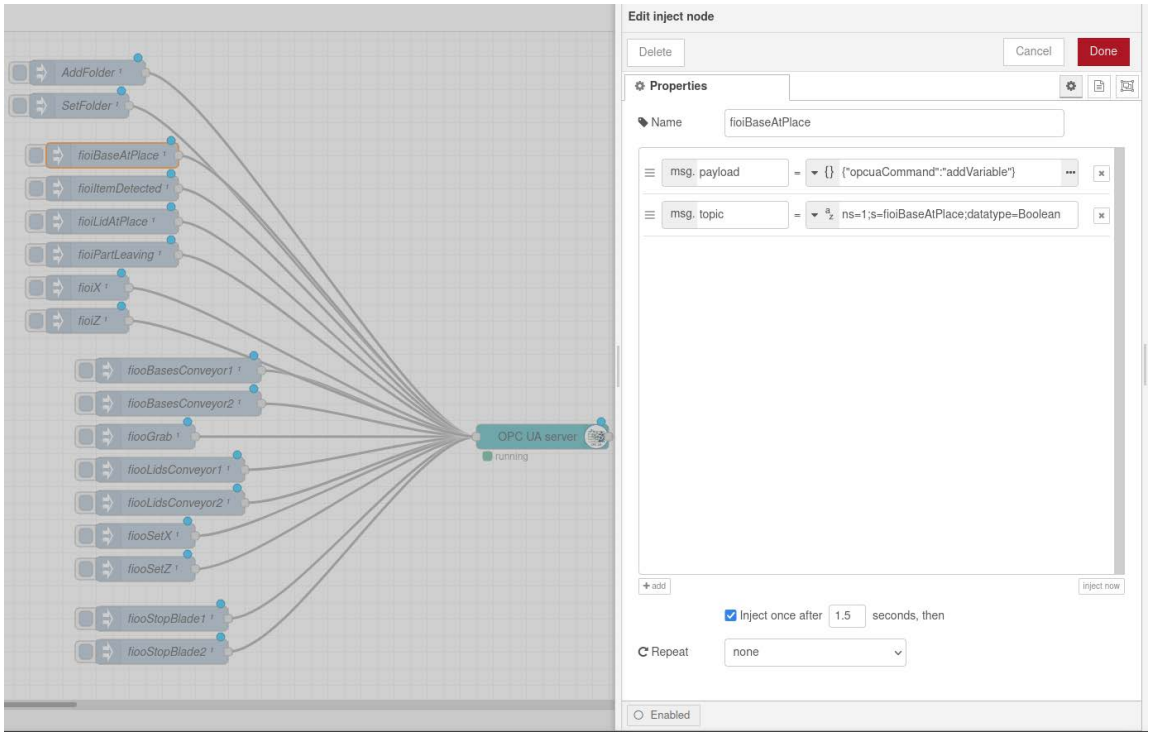


FIGURE 12. Graphical user interface of Node-RED.

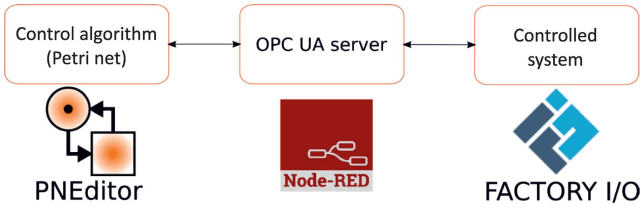


FIGURE 13. Graphical user interface of Node-RED.

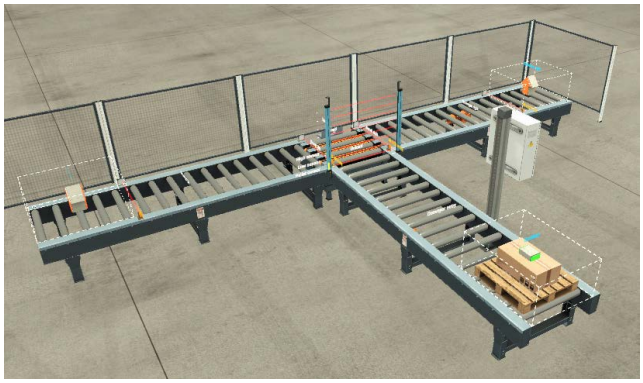


FIGURE 14. Factory I/O - Sorting by Height (Basic) scene.

UA server, into a deterministic state. Subsequently, automatic control takes place.

Starting the *Start* transition:

- starts the *Lids conveyor 1* and *Bases conveyor 1* conveyor belts, of which the *Lids conveyor 1* conveyor contains the top of the two-piece component and the *Bases conveyor 1* conveyor contains the bottom of the two-piece component
- moves the arm to the correct position in which it begins its operation,
- and also brings the stopping blades of both conveyors to the upper position.

When the top part arrives under the suction cup of the arm, which is detected by the *Lid at place* light sensor, and then the value *false* is written to the *fioiLidAtPlace* variable, the conveyor carrying this part is stopped. The two-axis arm with suction cup will come to rest by automatically firing *Go down* transition, which causes the marking to be set at the *Z* place corresponding to the Z axis position value of the arm, to the height at which the *Item detected* sensor at the end of this arm detects the top of the two-piece part and initiates the suction cup mechanism to be turned on by firing *Grab* transition. Initiation of the *Grab* transition shall write a value indicating the attachment of the part to the place associated with the *fiooGrab* variable corresponding to the gripping mechanism.

The next *Go up above base* transition (which is coupled with the *GfiooGrab* variable) moves the arm by writing specific marking to the places corresponding to the X and Z coordinates above the base of the two-part component. If this bottom part is located there (indicated by the *GBase at place* light sensor and the corresponding *GfiooBaseAtPlace* variable), it places the top part of the component on top of it by appropriately incrementing the Z coordinate and releasing the suction mechanism.

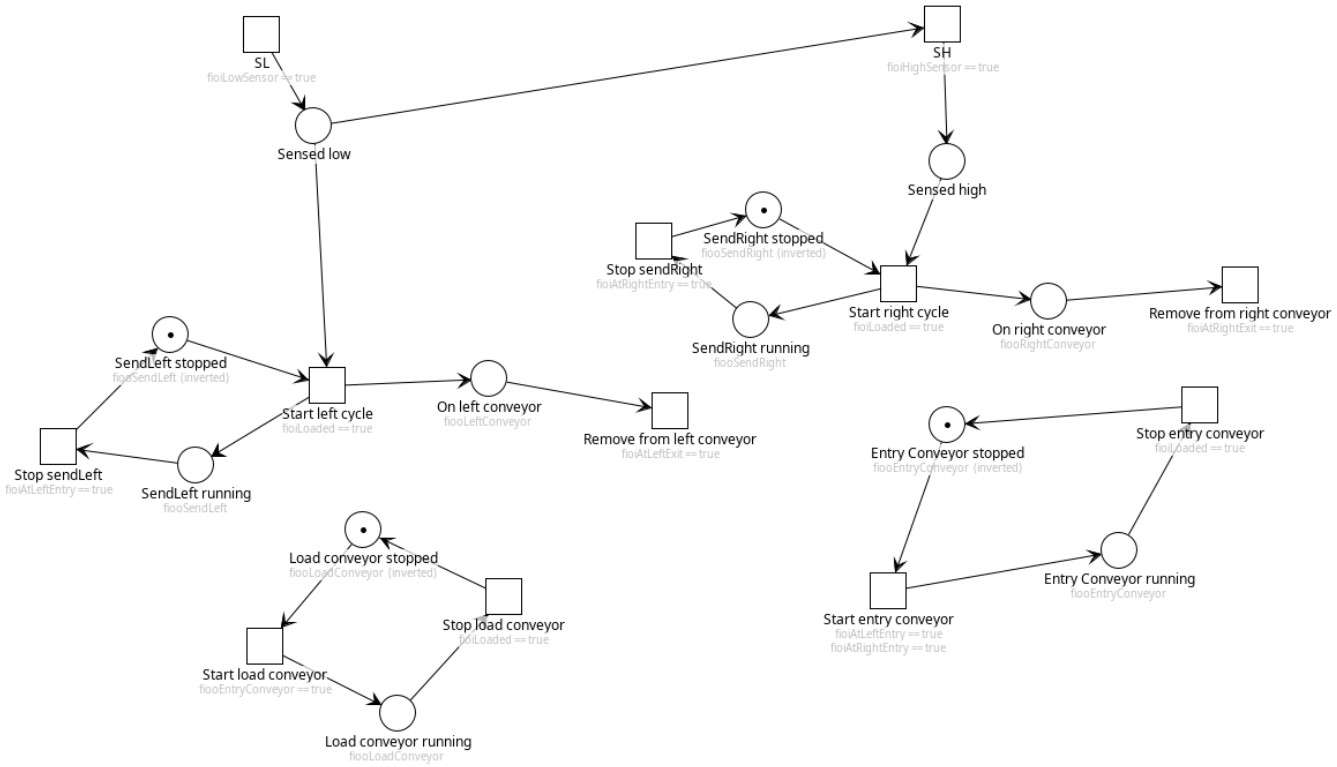


FIGURE 15. Petri net for control of scene Sorting by Height (Basic).

TABLE 6. Description of Petri net places for second case study - Sorting by Assembler (Analog) scene.

PLACE ID	PLACE DESCRIPTION	CONNECTION WITH OPC UA VARIABLE
Up	Represents the state when the arm has reached its topmost position.	-
BasesConv2	Represents the state of the second part of the bases conveyor (used for removing finished parts). Always on.	fiooBasesConveyor2
Blade1	Represents the state (up/down) of the *Stop Blade 1*, which is present on the lids conveyor. Always up.	fiooStopBlade1
Z	Represents the desired state of the variable bound to controlling the Z-coordinate of the manipulator arm, controlling its descend.	fiooSetZ
X	Represents the desired state of the variable bound to controlling the X-coordinate of the manipulator arm, controlling its lateral movement.	fiooSetX
LidsConv1	Represents the state (on/off) of the first (entry) part of the lids conveyor.	fiooLidsConveyor1
BasesConv1	Represents the state (on/off) of the first (entry) part of the bases conveyor.	fiooBasesConveyor1
Blade2	Represents the state (up/down) of the *Stop Blade 2*, which is present on the bases conveyor.	fiooStopBlade2
Above lid	Represents a state of the system when the arm is in a position above the lid.	-
Above base	Represents a state of the system when the arm is in a position above the base.	-
Grabbed	Represents whether the suction mechanism of the manipulator arm is engaged (marking >1).	fiooGrab

The release of the suction mechanism is combined in the Petri net with the automatic movement of the arm above the point at which it expects the next upper part, the setting of the stop blade (preventing the complete part from moving) to the lower position, and the lowering of the conveyor belts providing the pieces of the parts of the two-part part (in the *Remove finished and go up above lid* transition).

The last control step is to start the third conveyor belt, *Bases conveyor 2*, taking the completed part further.

The Petri net also contains *Above base* and *Above lid* places, which are not coupled with any of the OPC UA server's address space, and are only used for simplified monitoring of the model/Petri net's state.

We can conclude that the ability of hybrid systems control with our extended software application was successfully verified and can be generalised for other applications.

TABLE 7. Description of Petri net transitions for second case study - Sorting by Assembler (Analog) scene.

TRANSITION ID	TRANSITION DESCRIPTION	CONNECTION WITH OPC UA VARIABLE
Reached top	Used for sensing if the arm has reached its topmost position.	fioiZ == 0
Go down	Causes the manipulator arm to descend if the X coordinate is either above base or above lid.	fioiX == 1.105 , fioiX == 1.1 , fioiX == 8.85
Start	Used for setting all the required variables (place markings) to a predictable state and starting the process.	-
Stop	Stops the lids and bases conveyors when the parts reach a position where the arm can manipulate them based on sensor values.	fioiBaseAtPlace == true , fioiLidAtPlace == true
Part left	Once the *Part leaving* sensor is not intersected anymore, the Stop blade on the bases conveyor is set to its "upper" or "blocking" position.	fioiPartLeaving == false
Go up above base	Triggered together with setting the state of the suction mechanism at the end of the manipulator arm. Once the lid is grabbed, causes the arm to move above base by setting the required coordinates.	fiooGrab == true
Remove finished and go up above lid	Triggered by reaching a predefined depth and only if the arm is currently above base. Causes the Stop blade on the bases conveyor to be set to its "lower" or "non-blocking" position, starting the bases conveyor and also causing the manipulator arm to move above lid by setting the required coordinates.	fioiZ >= 8.86
Grab	In case the sensor at the end of the manipulator arm detects an item, the suction mechanism used to grab parts is enabled.	fioiItemDetected == true

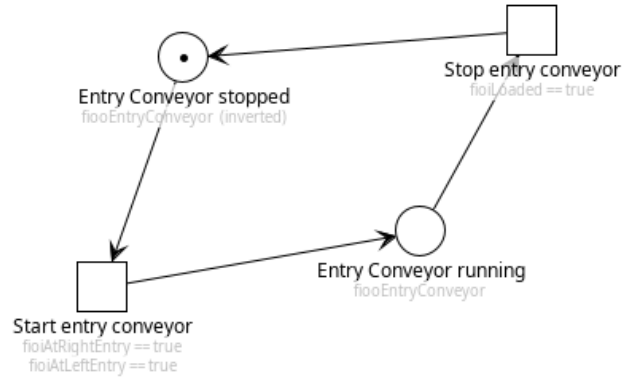


FIGURE 16. Petri net for control of scene Sorting by Height (Basic) - control of entry conveyor (detail).

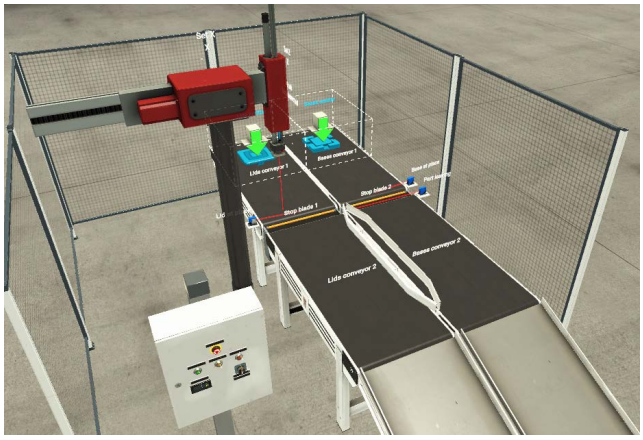


FIGURE 17. Factory I/O - Assembler (Analog) scene.

V. CONCLUSION

The article presents an extension of the PNEditor able to control systems using OPC UA communication protocol. This new software tool enables to control discrete event or hybrid systems using discrete, continuous or hybrid Petri nets thus supporting the control paradigm according to which the Petri net control logic is implemented in personal computer and control commands are sent using OPC UA. The main virtue of the upgraded software tool is its capability to control complex discrete event and hybrid systems exploiting the Petri nets formalism able to support many challenging scenarios in modern production systems operating in the Industry 4.0 framework.

Modelling systems using Petri nets has the huge advantage of being clear and easy to understand and visualise. Combining this feature with the ability to control in the manner described in this article is a significant simplification over commonly used control methods. This, and the architecture where the system is controlled directly from within the operating system environment, allows the "control algorithm" to be changed extremely quickly, and to react to changes in the system. This can be useful, for example, in the design and prototyping phase of a controlled system.

Petri nets provide a very clear and graphical way of designing a control algorithm. This is due to the fact that Petri nets can represent the state of a system by decomposing it into its individual sub-states. In practice, it may be necessary to optimise the control system. This requirement may arise because a production process needs to be optimised or the production process has changed. If the control system is programmed in a common text-based programming language, this change can take the programmer quite a long time. The graphical nature of Petri nets makes it relatively easy to change the control algorithm and put it into practice. There are also mathematical methods for analysing Petri nets that can be used for the purpose of determining whether a Petri net has been designed correctly.

As our research has documented, there is a lack of applications that enable automatic control by taking advantage of the formalism of Petri nets. As industrialization grows, so do the requirements for control capabilities, and it is important for commercial companies to be able to get systems up and running as quickly as possible, i.e., at the lowest possible cost.

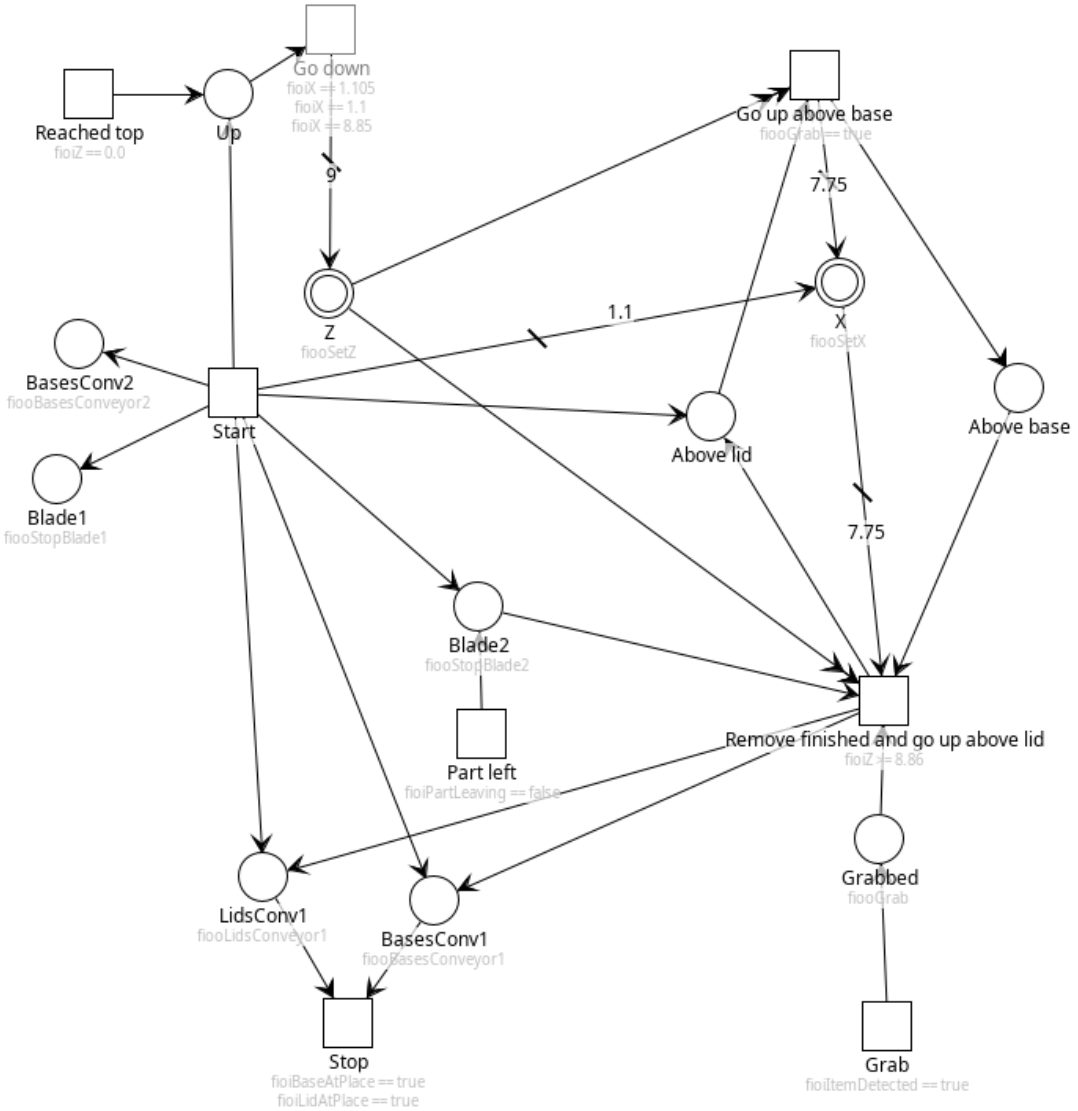


FIGURE 18. Petri net for control of scene Assembler (Analog).

We believe that the present approach can be a more suitable alternative in certain areas and applications than other control methods that are widely used nowadays.

The aim of this work was to implement an application that enables modelling of discrete, continuous and hybrid systems, and to enable their direct automated control via the OPC UA standard. This objective has been demonstrably fulfilled. The application has been developed with an emphasis on user-friendliness.

The application developed in this work can also serve as a teaching tool, as it extends the capabilities of the clear and simple editor by using continuous Petri net elements, and thus allows to easily demonstrate the usefulness of modelling with hybrid Petri nets, which was not possible before.

The application can be further extended and adapted. One possible enhancement is to extend the implementation to include the secure connection capabilities defined by the OPC UA standard, which are also supported by the Eclipse Milo library. Another potential enhancement that may be implemented in the future is the support of a so-called discovery server, i.e. the automatic discovery of OPC UA servers accessible in the network. There is also potential for enhancement in the user interface, for example, in better defining conditions for firing transitions - allowing the creation of conditions composed of the states of multiple variables.

There is also room for improvement in terms of extending the potential of Petri net formalism. For example, functionality that may prove useful in the course of using the application may include the ability to define the capacity of places in Petri net, the ability to define the timing of individual transitions, etc.

There are other challenges and opportunities to improve the developed application solution. It is possible to list some of them:

- If the system to be controlled is very complex, the Petri net will also be quite complex. In this case, it would be useful to develop functionalities for labeling (or logically grouping) the different components (subnets) of the net. Alternatively, it would be possible to add a comment to these groups. The bigger challenge is the implementation of hierarchical Petri nets for this purpose.
- Petri nets are based on a strong mathematical formalism. Various functionalities related to Petri net analysis (reachability of states, deadlock analysis, etc.) could be implemented in the tool.
- Petri net analysis tools could support graphical output of those analyses in the form of matrices or graphs. Since several properties of the Petri net (e.g., coverability tree, P/T invariants) are useful to display in such a form.

The scientific and application contributions as declared in this section describe the developed original modelling and control procedures and solutions for discrete event systems, and can further be modified for the next research and practice in Industry 4.0.

Petri nets are a very versatile and insightful tool for modelling and control of discrete-event systems. Hybrid Petri nets, of which there are still only few applications for control, bring new and interesting possibilities for control not only discrete-event systems, but also systems that also have a continuous component. For the purpose of controlling manufacturing processes, a hybrid system model represents the behaviour of dynamical systems in which the states can change both continuously and instantly. Such systems develop as a result of the inherent dynamics or when control algorithms involving digital smart devices are applied to continuous-time systems (e.g. mechatronic systems with bumps, electrical and mechanical devices for switching control). Due to the interaction between digital and analog parts of a complex manufacturing system, hybrid dynamics may be unavoidable, and hybrid control may be used to improve performance and robustness compared to conventional control (PID).

SUPPLEMENTARY MATERIAL

Video examples can be found here: https://www.youtube.com/watch?v=SHjZ7VDZ8E0 and https://www.youtube.com/watch?v=U33glvqCctw.

ACKNOWLEDGMENT

The authors would like to thank to Matej Marton for proposing the design concept, programming the implementation, and his all-round support.

REFERENCES

[1] L. J. Planke, Y. Lim, A. Gardi, R. Sabatini, T. Kistan, and N. Ezer, "A cyber-physical-human system for one-to-many UAS operations: Cognitive load analysis," *Sensors*, vol. 20, no. 19, p. 5467, Sep. 2020.

[2] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*, vol. 1. Berlin, Germany: Springer, 2010.

[3] H.-S. Chiang, M.-Y. Chen, and Y.-J. Huang, "Wavelet-based EEG processing for epilepsy detection using fuzzy entropy and associative Petri net," *IEEE Access*, vol. 7, pp. 103255–103262, 2019, doi: 10.1109/ACCESS.2019.2929266.

[4] R. Wiśniewski, G. Bazydło, P. Szcześniak, and M. Wojnakowski, "Petri net-based specification of cyber-physical systems oriented to control direct matrix converters with space vector modulation," *IEEE Access*, vol. 7, pp. 23407–23420, 2019, doi: 10.1109/ACCESS.2019.2899316.

[5] Z. Hajduk and J. Wojtowicz, "FPGA implementation of fuzzy interpreted Petri net," *IEEE Access*, vol. 8, pp. 61442–61452, 2020, doi: 10.1109/ACCESS.2020.2983276.

[6] R. Wiśniewski, "Dynamic partial reconfiguration of concurrent control systems specified by Petri nets and implemented in Xilinx FPGA devices," *IEEE Access*, vol. 6, pp. 32376–32391, 2018, doi: 10.1109/ACCESS.2018.2836858.

[7] M. Bashir, J. Zhou, and B. B. Muhammad, "Optimal supervisory control for flexible manufacturing systems model with Petri nets: A place-transition control," *IEEE Access*, vol. 9, pp. 58566–58578, 2021, doi: 10.1109/ACCESS.2021.3072892.

[8] C. Blume, S. Blume, S. Thiede, and C. Herrmann, "Data-driven digital twins for technical building services operation in factories: A cooling tower case study," *J. Manuf. Mater. Process.*, vol. 4, no. 4, p. 97, Sep. 2020.

[9] H. Kaid, A. Al-Ahmari, Z. Li, and R. Davidrajuh, "Intelligent colored token Petri nets for modeling, control, and validation of dynamic changes in reconfigurable manufacturing systems," *Processes*, vol. 8, no. 3, p. 358, Mar. 2020.

[10] I. Pombo, L. Godino, J. A. Sánchez, and R. Lizarralde, "Expectations and limitations of cyber-physical systems (CPS) for advanced manufacturing: A view from the grinding industry," *Future Internet*, vol. 12, no. 9, p. 159, Sep. 2020.

[11] A. Giua; C. Seatzu; F. Sessego, "Simulation and analysis of hybrid Petri nets using the MATLAB tool HYPENS," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2008, pp. 1922–1928.

[12] J. Julvez, C. Mahulea, and C.-R. Vazquez, "Analysis and simulation of manufacturing systems using SimHPN toolbox," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Aug. 2011, pp. 432–437.

[13] M. Dotoli, M. P. Fanti, and G. Iacobellis, "A freeway traffic control model by first order hybrid Petri nets," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Aug. 2011, pp. 425–431.

[14] M. P. Fanti, G. Iacobellis, A. M. Mangini, and W. Ukovich, "Freeway traffic modeling and control in a first-order hybrid Petri net framework," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 90–102, Jan. 2014.

[15] M. Dotoli, M. P. Fanti, and A. M. Mangini, "Fault monitoring of automated manufacturing systems by first order hybrid Petri nets," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, Aug. 2008, pp. 181–186.

[16] N. Costantino, M. Dotoli, M. Falagario, M. P. Fanti, and A. M. Mangini, "A model for supply management of agile manufacturing supply chains," *Int. J. Prod. Econ.*, vol. 135, no. 1, pp. 451–457, Jan. 2012.

[17] E. Kučera, O. Haffner, P. Drahoš, J. Cigánek, J. Štefanovič, and Š. Kozák, "New software tool for modelling and control of discrete-event and hybrid systems using Petri nets," *Comput. Informat.*, vol. 39, no. 3, pp. 568–586, 2020.

[18] A. C. Gaona, J. M. Chavez, and C. R. Vazquez, "RCPetri: A MATLAB app for the synthesis of Petri net regulation controllers for industrial automation," in *Proc. 26th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2021, pp. 1–7, doi: 10.1109/ETFA45728.2021.9613441.

[19] R. Nunes, L. Gomes, and J. P. Barros, "A graphical editor for the input-output place-transition Petri net class," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom. (EFTA)*, Sep. 2007, pp. 788–791, doi: 10.1109/EFTA.2007.4416858.

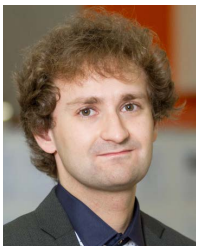
[20] M. A. Drighiciu, G. Manolea, D. C. Cismaru, and A. Petrisor, "Hybrid Petri nets as a new formalism for modeling electrical drives," in *Proc. Int. Symp. Power Electron., Electr. Drives, Autom. Motion*, Jun. 2008, pp. 626–631, doi: 10.1109/SPEEDHAM.2008.4581168.

[21] R. Davidrajuh, "Revisiting Petri net modeling of the cigarette Smokers' problem: A GPenSIM approach," in *Proc. Eur. Model. Symp.*, Nov. 2013, pp. 195–200, doi: 10.1109/EMS.2013.34.

[22] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, nos. 3–4, pp. 213–254, 2007.

[23] A. L. Dias, G. S. Sestito, A. C. Turcato, and D. Brandao, "Panorama, challenges and opportunities in PROFINET protocol research," in *Proc. 13th IEEE Int. Conf. Ind. Appl. (INDUSCON)*, Nov. 2018, pp. 186–193, doi: 10.1109/INDUSCON.2018.8627173.

[24] L. Zhang and N. Xie, "Research of Ethernet/IP and development of its network node," in *Proc. 2nd Int. Conf. Consum. Electron., Commun. Netw. (CECNet)*, Apr. 2012, pp. 486–489, doi: 10.1109/CECNet.2012.6201680.
[25] S. Chen, C.-L. Li, S.-C. Han, and F. Pan, "The design and implementation of Modbus/TCP communication on WinCE platform," in *Proc. 30th Chin. Control Conf.*, 2011, pp. 4710–4713.
[26] V. Q. Nguyen and J. W. Jeon, "EtherCAT network latency analysis," in *Proc. Int. Conf. Comput., Commun. Autom. (ICCCA)*, Apr. 2016, pp. 432–436, doi: 10.1109/CCAA.2016.7813815.
[27] T. Hannelius, M. Salmenpera, and S. Kuikka, "Roadmap to adopting OPC UA," in *Proc. 6th IEEE Int. Conf. Ind. Inform.*, Jul. 2008, pp. 756–761.
[28] B. M. Wilamowski and J. D. Irwin, *Industrial Communication Systems*. Boca Raton, FL, USA: CRC Press, 2018.
[29] L. Zheng and H. Nakagawa, "OPC (OLE for process control) specification and its developments," in *Proc. 41st SICE Annu. Conf.*, 2002, pp. 917–920, doi: 10.1109/SICE.2002.1195286.
[30] S. Cavalieri, "A proposal to improve interoperability in the Industry 4.0 based on the open platform communications unified architecture standard," *Computers*, vol. 10, no. 6, p. 70, May 2021, doi: 10.3390/computers10060070.
[31] A. Eckhardt, S. Müller, and L. Leurs, "An evaluation of the applicability of OPC UA publish subscribe on factory automation use cases," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2018, pp. 1071–1074, doi: 10.1109/ETFA.2018.8502445.
[32] C. P. Iatrou and L. Urbas, "Efficient OPC UA binary encoding considerations for embedded devices," in *Proc. IEEE 14th Int. Conf. Ind. Informat. (INDIN)*, Jul. 2016, pp. 1148–1153, doi: 10.1109/INDIN.2016.7819339.
[33] K. Mandittereza. (2021). *OPC UA Technology Mapping: Data Encoding, Data Security and Transport Protocols*. Accessed: Aug. 3, 2022. [Online]. Available: https://www.youtube.com/watch?v=zYjKK0F3RDg
[34] N. J. Dingle, W. J. Knottenbelt, and T. Suto, "PIPE2: A tool for the performance evaluation of generalised stochastic Petri nets," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 34–39, 2009.
[35] P. Bonet, C. M. Lladó, R. Puijaner, and W. J. Knottenbelt, "PIPE v2. 5: A Petri net tool for performance modelling," in *Proc. 23rd Latin Amer. Conf. Inform. (CLEI)*, Oct. 2007. [Online]. Available: https://sarahtattersall.github.io/PIPE/
[36] O. Kummer, F. Wienberg, M. Duvigneau, M. Köhler, D. Moldt, and H. Rölke, "Renew-the reference net workshop," in *Proc. 21st Int. Conf. Appl. Petri Nets Tool Demonstrations*. Aarhus, Denmark: Aarhus University, 2000, pp. 87–89.
[37] A. Ratzer, "CPN tools for editing, simulating, and analysing coloured Petri nets," in *Proc. Int. Conf. Appl. Petri Nets*. Berlin, Germany: Springer, Jun. 2003, pp. 450–462.
[38] M. Riesz, M. Seckár, and G. Juhás, "PetriFlow: A Petri net based framework for modelling and control of workflow processes," in *Proc. ACSD/Petri Nets Workshops*, 2010, pp. 191–205.
[39] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. London, U.K.: Pearson, 1995.
[40] H. Alla and R. David, "Continuous and hybrid Petri nets," *J. Circuits, Syst., Comput.*, vol. 8, no. 1, pp. 159–188, 1998.
[41] *Eclipse Milo*. Accessed: Aug. 3, 2022. [Online]. Available: https://projects.eclipse.org/projects/iot.milo
[42] *Simple OPC UA Stack Benchmarking Results*. Accessed: Aug. 3, 2022. [Online]. Available: https://github.com/kevinherron/stack-bench#results
[43] *Eclipse Public License—V 2.0*. Accessed: Aug. 3, 2022. [Online]. Available: https://www.eclipse.org/legal/epl-2.0/
[44] H. Böck, "IntelliJ IDEA and the NetBeans platform," in *Platform*, vol. 7. New York, NY, USA: Apress, 2012, pp. 431–437.
[45] B. Riera and B. Vigário, "HOME I/O and FACTORY I/O: A virtual house and a virtual plant for control education," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9144–9149, Jul. 2017.
[46] *OPC UA Servers*. Accessed: Aug. 3, 2022. [Online]. Available: https://www.unified-automation.com/downloads/opc-ua-servers.html
[47] *OPC UA Server Implementation*. Accessed: Aug. 3, 2022. [Online]. Available: https://github.com/eclipse/milo/blob/master/milo-examples/
[48] *About: Node-RED*. Accessed: Aug. 3, 2022. [Online]. Available: https://nodered.org/about/
[49] W. P. Stevens, "How data flow can improve application development productivity," *IBM Syst. J.*, vol. 21, no. 2, pp. 162–178, 1982.



ERIK KUČERA received the graduate degree from the Faculty of Electrical Engineering, Slovak University of Technology in Bratislava (FEI STU), Slovakia, in 2013, and the Ph.D. degree in mechatronic systems, in 2016.

He is with the Faculty of Electrical Engineering and Information Technology, Institute of Automotive Mechatronics, Slovak University of Technology in Bratislava. His focus is mainly on modern information and communication technologies and their use in the context of fourth industrial revolution Industry 4.0. His research interests include the Internet of Things, virtual and mixed reality, cloud computing, and new microcontrollers.

OTO HAFFNER received the graduate degree from the Faculty of Electrical Engineering, Slovak University of Technology in Bratislava (FEI STU), Slovakia, in 2013, and the Ph.D. degree in mechatronic systems, in 2016.

He is with the Faculty of Electrical Engineering and Information Technology, Institute of Automotive Mechatronics, Slovak University of Technology in Bratislava. His focus is mainly on modern machine vision methods and their use in the context of fourth industrial revolution Industry 4.0. His research interests include artificial intelligence and deep learning.

PETER DRAHOŠ received the graduate degree from the Faculty of Electrical Engineering, Slovak University of Technology in Bratislava (FEI STU), in 1985, and the Ph.D. degree in automation and control, in 2003.

Since 2012, he has been with the Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, where he is currently an Associate Professor. His main research interests include smart material actuators, sensors and automatic control, and industrial communication systems.

ALENA KOZÁKOVÁ received the graduate degree from the Faculty of Electrical Engineering, Slovak University of Technology in Bratislava (FEI STU), in 1985, and the Ph.D. degree in technical cybernetics, in 1996.

Since 2013, she has been with the Faculty of Electrical Engineering and Information Technology, Institute of Automotive Mechatronics, Slovak University of Technology in Bratislava, where she is currently a Professor. She is one of the leading experts in automatic control theory in Slovakia. Her specialty is mainly in the field of optimal control.