Hindawi
Scientific Programming
Volume 2019, Article ID 2604148, 17 pages
https://doi.org/10.1155/2019/2604148

Research Article

Facilitating the Quantitative Analysis of Complex Events through a Computational Intelligence Model-Driven Tool

Gregorio Díaz,[1] **Hermenegilda Macià**,[1] **Valentín Valero**,[1] **Juan Boubeta-Puig**,[2] **and Guadalupe Ortiz**[2]

[1]*School of Computer Science, University of Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain*
[2]*Department of Computer Science and Engineering, University of Cádiz, Avda. de La Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain*

Correspondence should be addressed to Gregorio Díaz; gregorio.diaz@uclm.es

Received 15 February 2019; Revised 10 May 2019; Accepted 2 July 2019; Published 29 July 2019

Guest Editor: Alvaro Rubio-Largo

Copyright © 2019 Gregorio Díaz et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Complex event processing (CEP) is a computational intelligence technology capable of analyzing big data streams for event pattern recognition in real time. In particular, this technology is vastly useful for analyzing multicriteria conditions in a pattern, which will trigger alerts (complex events) upon their fulfillment. However, one of the main challenges to be faced by CEP is how to define the quantitative analysis to be performed in response to the produced complex events. In this paper, we propose the use of the MEdit4CEP-CPN model-driven tool as a solution for conducting such quantitative analysis of events of interest for an application domain, without requiring knowledge of any scientific programming language for implementing the pattern conditions. Precisely, MEdit4CEP-CPN facilitates domain experts to graphically model event patterns, transform them into a Prioritized Colored Petri Net (PCPN) model, modify its initial marking depending on the application scenario, and make the quantitative analysis through the simulation and monitor capabilities provided by CPN tools.

1. Introduction

Complex event processing (CEP) [1] is a computational intelligence technology used to analyze and correlate big data streams in order to detect situations of interest in real time. Events can be generated from other ones by matching the so-called *event patterns*, which are templates describing the conditions to be met to recognize such situations.

CEP fits in event-driven service-oriented architectures (SOA 2.0), in which the communications between applications and services are conducted by the complex events produced upon pattern detection. The event patterns to be detected for a particular application domain are implemented by using event-processing languages (EPLs) [2]. The MEdit4CEP approach [3] was proposed to help domain experts with this implementation. This framework provides an editor with graphical modeling capabilities for easily specifying the CEP domain, event patterns and action definitions. This approach generates Esper EPL code [4] from the graphical models. Moreover, MEdit4CEP was extended by using the Prioritized Colored Petri Net (PCPN) formalism [5], and the new version was called MEdit4CEP-CPN [6, 7]. In this framework, graphic event pattern models are automatically transferred into its graphical PCPN counterpart, which is transferred again into a compatible PCPN code interpretable by CPN Tools [8] to execute the event patterns, with the aim to perform the semantic validation of them. Thus, this approach offers the advantage of providing a graphical model to perform such a validation: users benefit from a better understanding of the internal semantics of the model through visual simulation of the model, where they can see its dynamic behavior and locate possible errors, for instance, in a certain condition, operation, time window, etc. Therefore, users not only observe whether there are differences between expected and actual outputs, but they can also inspect the internal dynamics of pattern execution when these differences occur.

Although CEP is so powerful for recognizing complex events in real-time big data streams, domain experts must face how to define the quantitative analysis to be performed in response to the produced complex events. These quantitative analysis studies are carried out to evaluate existing or planned scenarios, to compare alternative scenarios or to find an optimal scenario. In particular, we will focus on time-dependent scenarios since CEP systems consist of event streams flowing in time.

The main aim of this paper is, therefore, to demonstrate how MEdit4CEP-CPN can be used for conducting such quantitative analysis of events of interest for an application domain, without requiring knowledge of any scientific programming language for implementing the pattern conditions. Thereby, end users are provided with an all-in-one tool for graphically modeling event patterns, transforming them into a PCPN model, modifying its initial marking depending on the application scenario, and making the quantitative analysis through the monitor capabilities provided by CPN Tools. Obviously, the use of CPN Tools requires some knowledge from users in order to conduct the quantitative analysis, at least for modifying the initial marking of the produced CPN model and then executing the simulations to obtain the results. As indicated in our plans for future work, we intend to alleviate this problem by enriching our graphical model for event pattern design in order to be able to set the initial conditions (event flow) at design time and adding the option to automatically execute the produced CPN. The obtained output would then be transformed into the corresponding complex events in the output flow.

The quantitative analysis will be done by simulation and will involve statistical investigation of output data (complex events), exploration of large data sets, proper visualization of those output data, and the validation of simulation experiments. The outputs obtained from the simulations depend on the input data (simple events) that feed the system. These input data are set up stochastically, according to a specific scenario model. Thus, appropriate statistical techniques can be used for both designing and interpreting simulation experiments.

The structure of the paper is as follows. Section 2 depicts a general background describing the technologies and tools used in this work. Section 3 specifies the different steps followed in this work to perform the quantitative analysis. A case study about the sick building syndrome is then presented in Section 4, with a quantitative analysis using our methodology. Section 5 presents the related works, and a comparative study with our framework. Finally, Section 6 presents the conclusions and lines of future work.

2. Background

In this section, we introduce the main technologies used in this work, CEP and Colored Petri Nets (CPNs), and a brief description of the MEdit4CEP-CPN tool.

2.1. Complex Event Processing. CEP is a technology that captures, analyzes, and correlates large amounts of simple events with the ultimate goal of detecting relevant or situations of interest in a particular domain. Captured events are data that can flow through information systems, be provided by devices such as sensors, or come from social networks, among others. Such data are called simple events because they are characterized by being mainly raw data. The possibility of processing such simple events will allow us to infer information with a greater degree of semantic knowledge, thus obtaining the so-called complex events. For instance, for a stockbroker, the fact that the shares of a certain company fall 2% may be insignificant; however, that in a short period of time, both the shares fall 2% and also news about the low solvency of the company are published in a newspaper of economy; it could mean that it would be appropriate to sell the shares as soon as possible. In this case, the simple events would be that within a period of time t, the shares of the company x drop at least 2% and there is a negative news about the company in economic press. The complex event would be the recommendation of immediate sale. This way, in a given context, we will be able to detect the situations that are specifically relevant in that context or application domain. In order to do this, as Figure 1 shows, it will be necessary to previously define a series of event patterns specifying the conditions that simple input events must satisfy to detect such a situation. These patterns are defined and deployed in a CEP engine—software used to match these patterns on the incoming event flows, capable of analyzing the data and providing situations of interest detected in real time. The main advantage of CEP compared to other traditional event analysis software is the added ability to process large amounts of data and notify situations of interest detected in real time, allowing reduction of considerably the latency in decision making. This decision making capability relies on the architecture where the CEP engine is deployed, allowing the system either to perform certain actions or to enact certain measures.

As previously mentioned, CEP is a technology that can be very useful in several application domains such as financial systems, health care, energy optimization, online sales and marketing, business intelligence, security, and transportation, among many others, since CEP objective is to offer a general paradigm to be applied to a great variety of systems [1, 9–14]. However, a deep knowledge of the application domain is required to be able to define the patterns that may be relevant to that domain depending on the simple events that can be obtained in the system. Companies have normally domain experts and computer science experts, but these skills usually do not fall on the same person and the definition of patterns in the language provided by CEP engines is not trivial. For this reason, in the past, we proposed MEdit4CEP [3].

MEdit4CEP was defined and implemented for the purpose of providing a tool for CEP pattern definitions appropriate for domain experts with no particular programming skills. Thus, MEdit4CEP is a model-driven solution for real-time decision making in SOA 2.0 that provides a graphical modeling editor for CEP domain definition and a graphical modeling editor for event pattern

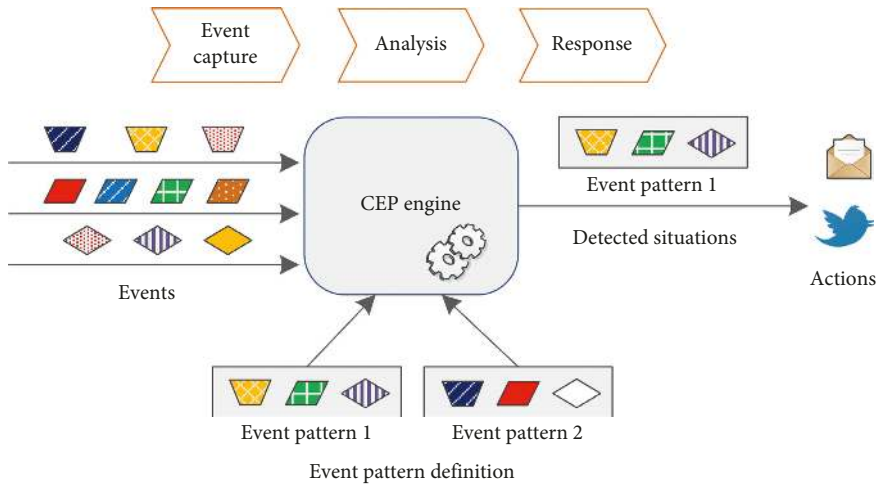


Figure 1: Complex event-processing stages.

definition, as well as automatic code generation and deployment from the patterns modeled by the domain expert.

2.2. Petri Nets and Quantitative Analysis. A Petri Net (PN) is defined as a bipartite-directed graph which has two types of nodes, places (depicted as circles) and transitions (depicted as rectangles), connected using arcs between either places and transitions (pt-arcs) or transitions and places (tp-arcs) [15]. Places of a Petri Net are used to represent system states and conditions, and a transition represents an action or an event producing a change in the system state.

Definition 1 (Petri Net). A Petri Net is a triple (P, T, F), where P is the set of places, T is the set of transitions, $X = P \cup T$ is the set of nodes, and $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs. For any node $x \in X$ (place or transition), we define the preconditions and postconditions of x, denoted by ${}^\bullet x$ and $x^\bullet$, respectively, as follows: ${}^\bullet x = \{y \in X \mid (y, x) \in F\}$, $x^\bullet = \{y \in X \mid (x, y) \in F\}$.

The dynamic evolution of a PN is captured by the so-called *markings*. A marking of a PN is a function $M: P \longrightarrow \mathrm{IN}$, which assigns a natural number to each place. This number is usually indicated as a set of dots inside the place or by a number beside the place, and it is called the number of *tokens* on the place. This number can be used for instance to indicate the number of events that must be processed or the number of processes in a queue.

A natural number (*arc weight*) is used to label a pt-arc. This number indicates how many tokens are required to fire (*execute*) an outgoing transition, by default, one. Tp-arcs also have a weight associated, which specifies how many tokens will be produced at the outgoing place when the transition is executed. A transition t can then be fired (*enabling condition*) when all its precondition places have at least as many tokens as the weight of the arc that connects them to t. The firing of a transition t removes from its precondition places a number of tokens equal to the weight of the pt-arc (p, t) connecting them and inserts new tokens on its postcondition places, according to the weights of those tp-arcs.

Petri nets are untimed and tokens do not carry any information. CPN [16] is a PN extension which incorporates data and time, allowing to model complex data structures attached to tokens. Thus, places have an associated *color set* (a data type), indicating the set of permitted token colors at a given place.

CPN Tools [8] is a widely used tool that allows us to create, edit, simulate, and analyze CPNs. The notation described below is the one used in this tool. In CPNs, we can have places with no attached information (color set *UNIT*), as in the plain model. But we can have other color sets, such as the set of integer numbers *INT*, a Cartesian product of two or more color sets as $INT2 = INT \times INT$, $INT3 = INT \times INT \times INT$, and a string ($STRING$). Each token has then an attached data value *(color)*, which belongs to the corresponding place color set.

In CPN Tools, the current number of tokens on every place is drawn in green beside the place circle, and the specific colors of these tokens are indicated using the notation $m'v$, meaning that we have m instances of color v. When we have several tokens on a place with different values, we use the symbol "++" to represent the union of them.

Arcs can have inscriptions (*arc expressions*), constructed using variables, constants, operators, and functions, whose evaluation matches the *color set* of the attached place. For a transition t with variables $y1, y2, \ldots$ in its input arc expressions, a *binding* of t is an assignment of specific values to each of these variables. A transition t is then *binding enabled* if there is a *binding* such that the evaluation of each input arc expression of t matches the corresponding tokens (with the same values) in the corresponding input place.

We can have guards associated to transitions, which can be used to restrict their firing. Guards are predicates constructed by using the variables, constants, operators, and functions of the model. For a guarded transition to be *fireable*, the evaluation of the guard must be true with the selected binding. A priority can also be associated to a transition. When two or more transitions can be fired (executed) at a given time, the transition with the highest

level of priority is fired first. A CPN with priorities is called a prioritized CPN (PCPN). Specifically, we use the following priorities: P_HIGH, P_NORMAL, P_LOW, P_LOW_1, P_LOW_2, ... P_LOW_n (for a certain $n \in \mathrm{IN}$) and P_MIN, following this decreasing order of priority.

CPN Tools allows us to split the model into pages, which is a useful feature to deal with large models. In this case, both substitution transitions and fusion places can be used to conform the whole model. Substitution transitions allow us to create hierarchical models, in which some transitions represent the actions enclosed in other CPN pages, while fusion places are places that are used in different pages, i.e., the places identified by a same fusion label are functionally the same place.

Example 1. Let us consider the PCPN depicted in Figure 2. Places *InputEv* and *OutEv* have *INT2* as color set and *ProcSqn* and *SqOut* have *INT* as color set. The initial marking of both places *ProcSqn* and *SqOut* is $1'1$ (one integer token with value 1). Place *InputEv* represents a flow of input events of a system that must process these events producing as output a flow with those events whose value is greater than 2 (place *OutEv*). Each event is represented by a Cartesian product of 2 integers (colorset *INT2*), in which the first component stands for the event number and the second component for the event value (integer). The initial marking of place *InputEv* is $M = 1'(1,1) + +1'(2,4) + +1'(3,0) + +1'(4,9)$, as shown in the figure. Transitions are labeled with their associated guard and priority information (P_NORMAL if empty), and arcs are labeled with their corresponding expressions. All the variables used in the expressions (n, m, v, k) are integers.

In this PCPN, place *SqOut* allows us to number sequentially the events produced on *OutEv*. Place *ProcSqn* acts as a sequence counter so as to process the event tokens on *InputEv* in order. Transition *selcond* must be fired when we have one token (n, v) on *InputEv* with a sequence number n equal to that indicated on *ProcSqn* that fulfills the condition $v > 2$. Transition *selcond* updates the sequence number on *ProcSqn*, by increasing it by one unit. Otherwise, when transition *selcond* cannot be fired, transition *incr_sq* is fired in order to increase the sequence number on *ProcSqn*, but it will stop firing when the sequence number on *ProcSqn* is greater than the maximum sequence number on *InputEv*.

The final marking obtained on the place *OutEv* is therefore $M' = 1'(1,4) + +1'(2,9)$ for the initial marking indicated in the figure. The final marking on *ProcSqn* is $1'5$ and on *SqOut* is $1'3$, and place *InputEv* keeps its initial marking.

Quantitative analysis in CPNs allows us to obtain relevant performance indexes of the system modeled. For instance, this analysis is used to obtain average response times, throughput, queue lengths, etc. In our case, the quantitative analysis can be used both to validate the event patterns defined and also to obtain predictive information by feeding the system with different event scenarios. Quantitative analysis using CPNs is usually based on simulations in order to obtain the measures of interest for the modeled scenario. This simulation-based quantitative analysis is performed through a number of lengthy simulations of a CPN model, during which data are collected from the occurring binding elements, firing of transitions, and markings reached so as to obtain estimates of measures of interest; in our case, the expected outputs of the system. This information is gathered by repeating the same experiment (simulation) a number of times, using the replication capabilities of CPN Tools and then using the monitoring capabilities of CPN Tools to extract the relevant data from the simulations. Specifically, we use place content break point and data-collector monitors, which allow us to determine whether a place becomes marked and extracts numerical data during simulations, respectively. For instance, these monitors can be used to count the number of times a specific transition has been fired across a simulation, to extract the marking of some specific places of the CPN model or to obtain the first instant at which a specific transition was fired.

Thus, the quantitative analysis is basically based on the monitor and replication capabilities of CPN Tools, which provides us with a report and log files which can be analyzed by using other well-known statistical computing tools, such as *R*, Matlab, and SPSS.

As an illustration, let us consider the CPN depicted in Figure 2. It might be of interest to know how many times transition *selcond* is fired, i.e., the amount of tokens gathered at place *OutEv*. Figure 3(a) shows the binder tools palette of CPN Tools for monitoring purposes, and Figure 3(b) shows the monitor that has been defined to count the number of firings of transition *selcond*. In this case, as it can be seen from the example, the result obtained by applying the monitor was 2.

Experiments can then be produced for different scenarios by modifying the initial marking, which can also be randomly generated so as to produce synthetic scenarios. For instance, we could define a function *M_init* to produce a random initial marking with n tokens for place *InputEv* in Figure 2 using a discrete uniform distribution, as follows:

$$\begin{aligned}\mathrm{funM\_init}(n) = {} & \mathrm{if}\,(n = 0)\mathrm{then\ nil\ else}\ 1'(n,\ \mathrm{discrete}(1,6)) \\ & ++\mathrm{M\_init}(n-1).\end{aligned} \tag{1}$$

To reproduce the experiments, we can use the replica capabilities provided by CPN Tools. The following expression simulates m times this example:

$$\mathrm{CPN'Replications.nreplications\ m}. \tag{2}$$

The outputs obtained for these experiments using these stochastic initial markings can then be analyzed with the statistical computing tools mentioned above.

2.3. MEdit4CEP-CPN. As previously explained, this tool was introduced in [6] as an extension of MEdit4CEP [3] to deal with the semantic validation of the modeled patterns.

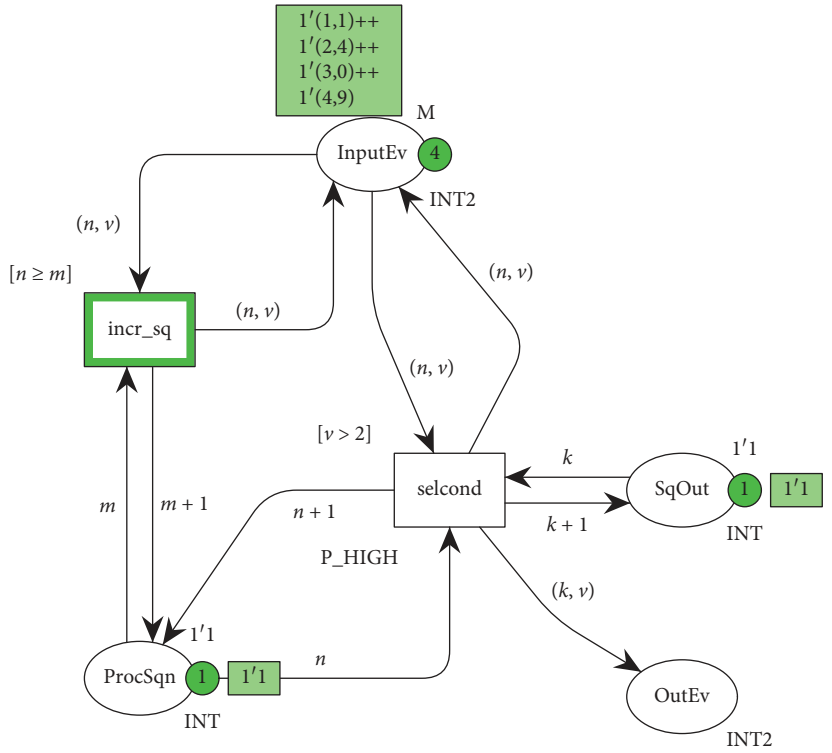
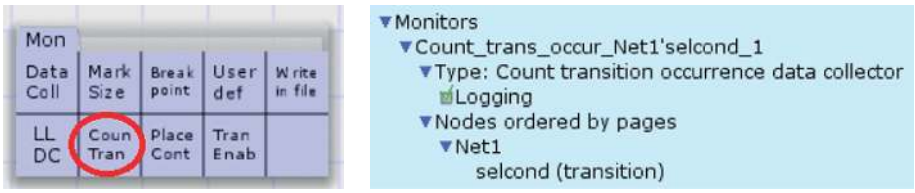


Figure 2: Graphical view of a marked PCPN.



(a) (b)

Figure 3: Monitor use in CPN Tools. (a) Monitor example. (b) Monitor example.

In particular, MEdit4CEP-CPN mainly consists of a domain-specific modeling language (DSL) and a graphical modeling editor for automatically transforming event pattern models into PCPN graphical models. Then, these models are validated and transformed into codes executable by Petri nets software.

This DSL was implemented using the Epsilon languages [17] for model-to-model transformation, model validation, and template-based code generation. Additionally, Epsilon EuGENia [18], a front-end for the graphical modeling framework, was used for implementing the editor. More details about the implementation can be found in [6].

Figure 4 illustrates the 7 phases, explained below, a user can follow to accomplish not only the semantic validation of the modeled patterns but also to be able to perform a quantitative analysis of the complex event properties in the studied scenarios.

2.3.1. Event Pattern Model Definition. In phase 1, the tool user is expected to graphically define the event patterns to be detected in a particular application domain.

2.3.2. Event Pattern Model Syntactic Validation. Once an event pattern has been modeled (phase 1), thanks to the use of the presented editor, the user can automatically validate the pattern syntax (phase 2). The editor will check whether the model conforms to the ModeL4CEP metamodel. Afterwards, the errors to be fixed before continuing will be shown. As of this phase, we can accomplish a semantic validation through PCPNs (phases 3, 4, 5, and 6); otherwise, phase 7 can be performed with the aim of automatically transforming the model into EPL code.

2.3.3. Event Pattern Model Transformation to PCPN Model. In phase 3, the event pattern models are automatically transformed into a PCPN model. In order to provide such a functionality, the editor has been provided with a metamodel for PCPN and a set of model-to-model transformation rules that we have defined and implemented for this purpose. Thus, a PCPN conforming to the named metamodel is generated.

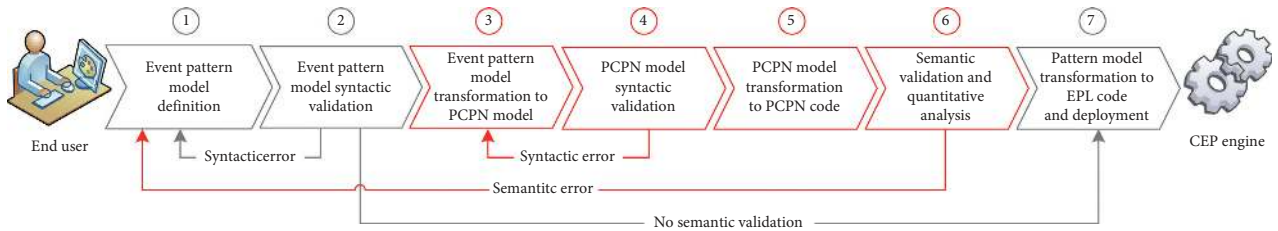


Figure 4: MEdit4CEP-CPN: a model-driven approach for CEP modeling by PCPN.

2.3.4. PCPN Model Syntactic Validation. Once the PCPN model has been automatically generated, in phase 4, domain experts may modify the PCPN according to their needs. For instance, they might be interested in editing the initial marking to check other particular scenarios of their interest. Then, after the PCPN edition, (1) it is checked whether the new model conforms to the PCPN metamodel and (2) whether the validation rules are satisfied through a syntactic validation. The errors that should be fixed before continuing with the following phase would be shown at this stage.

2.3.5. PCPN Model Transformation to PCPN Code. In phase 5, the PCPN model is automatically transformed into executable PCPN code (PCPN code refers to a proprietary PCPN file format that can be executed by a specific software); a set of model-to-text transformation rules have been defined and implemented for this purpose.

2.3.6. Semantic Validation and Quantitative Analysis. The expert in charge of simulating and analyzing the PCPN will then feed the net with an arbitrary number of initial markings (stream of events) in phase 6. This way, it will be possible to check if the event pattern is semantically correct, as well as a quantitative analysis will be carried out (see Section 3). In the case a semantic error is detected, we should return to phase 1.

2.3.7. Pattern Model Transformation to EPL Code and Deployment. Finally, in phase 7, the event pattern model is automatically transformed into EPL code and deployed in the CEP engine in question. In this work, we are generating code for the Esper CEP engine, but new transformation rules for other CEP engines of interest may be easily created and integrated in the proposed editor.

Therefore, we can conclude that, we have a top-down approach in which users can graphically define what they want to model (event patterns) and the proposed system automatically provides the implementation code. In this way, according to the capabilities associated to phases 5 and 6, MEdit4CEP-CPN allows us to infer additional meaningful information and to obtain predictive results about the analyzed pattern by feeding the system with different initial scenarios (markings).

3. Quantitative Analysis of Complex Events

CEP is a new class of event-processing solution which integrates into standard middleware architectures and enables event processing to be embedded in any standard enterprise application. This new service technology brings the power of event-driven insight into any industry and any end user. But sometimes, it is not easy to be used by domain experts. In this sense, MEdit4CEP-CPN was created to reach the next objectives in the CEP scope:

(i) Creation of user-friendly models for facilitating domain experts the task of defining event patterns

(ii) Syntactic and semantic model validation through model-driven techniques and CPN Tools, respectively, without requiring deep knowledge on CEP or PCPN formalism

(iii) Model-to-model and model-to-text transformations for automatically transforming the event pattern models into both PCPN models and EPL code and also the PCPN models into code executable by CPN Tools

(iv) Plug-in based solution, what will allow us to easily extend it with additional capabilities such as glass box and black box testing techniques for testing models, or predictive analytics for predicting future scenarios by analyzing the historical data

In this paper, we focus on phase 6 (semantic validation and quantitative analysis), briefly described in Section 2.3, which receives as an input the automatically generated PCPN code executable by CPN Tools. In this paper, more specifically, we define and carry out the particular phases that must be followed to conduct the quantitative analysis of the system under study (see Figure 5).

3.1. 6(a) Scenario Configuration. The initial marking (M_0) of the generated PCPN is initialized with an ordered flow of simple events ($F = e_1, e_2, e_3, \ldots$), representing a specific scenario. This event flow can be introduced manually or generated automatically by using deterministic or probability distribution functions provided by CPN Tools. Note that the automatic data generation is very convenient for analysis purpose.

3.2. 6(b) Deterministic Quantitative Analysis. The PCPN is then executed using CPN Tools in order to obtain the

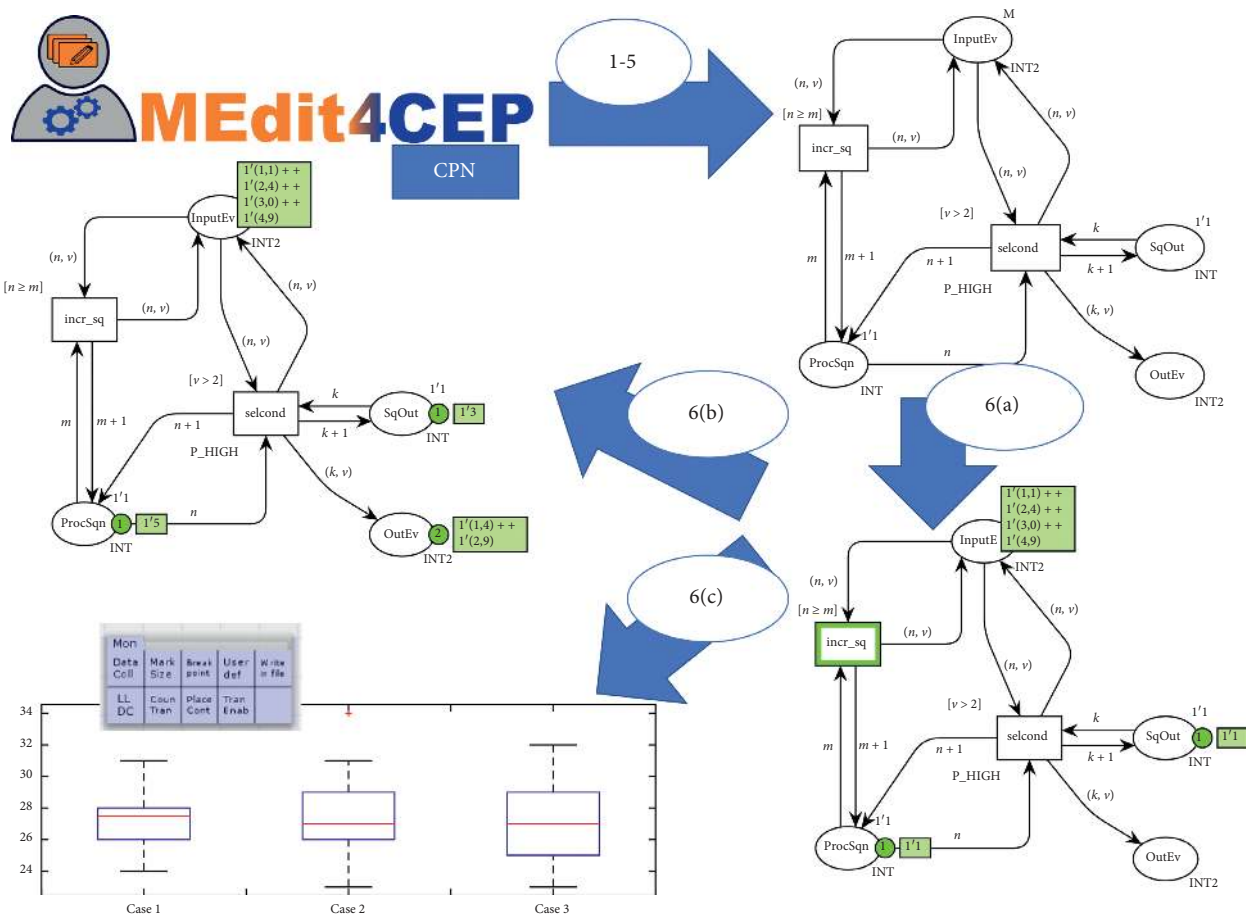


Figure 5: Phases to conduct the quantitative analysis.

corresponding output (detected complex events). Thus, this phase allows us to conduct the semantic validation as well as the quantitative analysis with a deterministic input event flow.

3.3. 6(c) Stochastic Quantitative Analysis. Alternatively to phase 6(b), the initial marking M_0 is obtained using stochastic input event flows, according to different scenarios, and the PCPN is then executed using simulations and the monitor capabilities provided by CPN Tools.

Therefore, the analysis of different scenarios is performed in phase 6, in which we feed the PCPN with initial markings that simulate the scenarios to be studied. In particular, we can produce the initial markings by using either deterministic functions that simulate specific situations or using probability distributions that produce stochastic values reproducing scenarios of interest. For instance, a normal distribution could be used to produce increments in a temperature measure, an exponential distribution to represent the users arriving at a hospital emergency room, and so on. In this case, when using stochastic initial markings, CPN Tools allows us to replicate the execution of a CPN by automatically producing the initial markings, and the monitor capabilities can then be used to collect the relevant data from the simulations.

4. Case Study: Sick Building Syndrome

As an illustration of the methodology described in the previous section, we consider a scenario of events related to the sick building syndrome (SBS), which is considered by the World Health Organization (WHO) [19] as a group of symptoms that people suffer in a building for no apparent reason. Some of the SBS symptoms are nose, throat, and eye irritation; itching, dry and red skin; dry mucous membranes sensation; mental fatigue; and headaches, and dizziness, and nausea. These symptoms tend to increase in severity as people spend more time in the building but get reduced over time or even disappear when people are away from the building. Apart from this health problem, people's work performance becomes obviously affected, with a corresponding loss of productivity. The SBS is widespread and may occur in hospitals, offices, apartment houses, nurseries, schools, and so on. Although the cause of SBS is unclear, some main factors related to indoor air quality (IAQ) are chemicals emissions from different sources, particles, radon, pets and pests, microbes, temperature, humidity, and ventilation.

In this sense, the WHO provides the guidelines [20] for the protection of public health from risks for some selected pollutants commonly present in indoor air, including the carbon monoxide, which is the pollutant that we consider in this work.

These are the recommendations related to indoor exposures of *CO*:

(i) 100 mg/m^3 for 15 minutes (assuming light exercise and that such exposure levels do not occur more often than one per day)

(ii) 35 mg/m^3 for 1 hour (assuming light exercise and that such exposure levels do not occur more often than one per day) (30 mg/m^3 is also recommended in some European Agencies such as https://www.anses.fr/fr/system/files/AIR2004etVG003Ra.pdf)

(iii) 10 mg/m^3 for 8 hours (arithmetic mean concentration and light-to-moderate exercise)

(iv) 7 mg/m^3 for 24 hours (arithmetic mean concentration, assuming that the exposure occurs when people are awake and alert, but not exercising)

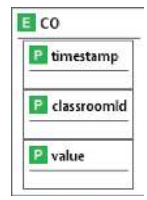
Let us consider, for instance, a school as the specific indoor scenario, where children stay in a classroom for 5 h/day. We then focus on the second recommendation: a nonhealthy indoor air quality corresponds to an average level greater than or equal to 35 mg/m^3 of *CO* for 1 hour.

Next, we follow the phases described in Section 2.3 together with the new phases proposed in this paper (see Section 3) in order to make the quantitative analysis of the complex events detected for this scenario by using MEdit4CEP-CPN and CPN Tools.

4.1. Phases 1–5: Event Pattern Model Definition, Syntactic Validation, and Transformation to PCPN Model, as well as PCPN Model Syntactic Validation and Transformation to PCPN Code. The domain considered for this hypothetical scenario consists of *CO* measurements gathered every 5 minutes in a specific classroom. Thus, a simple event consists of a measure for the *CO* pollutant, the classroom identifier where the measure was gathered and the timestamp of the measure. We consider the event time stamps as integers (in minutes), classrooms identifiers as strings, and *CO* values as real numbers. Using MEdit4CEP-CPN, we can easily define this domain (*CO* event type). Figure 6(a) depicts the domain modeled and syntactically validated with MEdit4CEP-CPN, and Figure 6(b) shows its automatic translation to EPL code.

Patterns can be easily modeled and syntactically validated using the tool. Figure 7(a) shows the *CO_Avg*-modeled pattern, which computes the *CO* pollutant average during the last hour, while Figure 7(b) shows its automatic transformation into EPL code. In the same way, Figure 8(a) shows the *CO_Unhealthy* modeled pattern, to detect and recognize situations in which the average computed with *CO_Avg* is greater than or equal to 35 mg/m^3, and Figure 8(b) shows its transformation into EPL code.

Using our MEdit4CPN-CPN tool again, the *CO_Avg* and *CO_Unhealthy* event patterns are automatically transformed into their corresponding PCPN models consisting of four different pages (see [6] for a complete description of these transformations): two pages for the pattern transformation to obtain the *CO* pollutant average and the condition to establish whether the threshold value of 35 mg/m^3 has been reached or not (see Figures 9 and 10, respectively) and two pages for simple and complex events (see Figures 11 and 12, respectively). The corresponding CPN Tools declarations are also shown in Listing 1.1. For simplicity, we consider in the PCPN model that one unit time corresponds to 5 minutes, therefore tp_CO_Avg = 12 in Listing 1.1.



(a)

```
create schema CO(timestamp integer,
  classroomId string, value double);
```

(b)

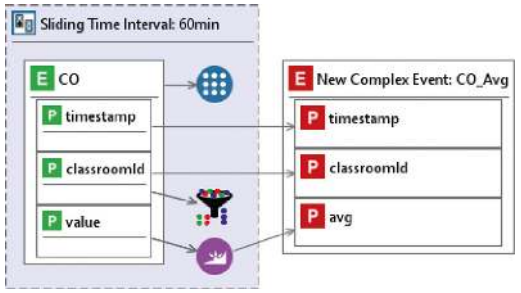
Figure 6: Modeled domain. (a) Domain in MEdit4CEPCPN. (b) Domain in EPL.

4.2. Phase 6(a): Scenario Configuration. Similarly to Example 1, the initial marking in place *CO_in* contains the tokens that represent the input events (see Figure 13). In this case, these events represent the *CO* values that have been measured. In the scenario considered, for the initial marking of the *CO_in* place, we start with a *CO* value of 3.0 mg/m^3. This value is then increased by a value of 0.5 mg/m^3 every 5 minutes.

4.3. Phase 6(b): Deterministic Quantitative Analysis. After executing the PCPN model by running a simulation, a value of 30.25 mg/m^3 is reached after 60 time units (300 minutes) in the *CO_Avg* place (see Figure 14), so the *CO_Unhealthy* place is empty, because no complex event has been produced corresponding to this situation. As a consequence, in this scenario and with the values produced by this specific simulation, the threshold value of 35 mg/m^3 was not reached as the average level of *CO* computed in the considered period of time.

4.4. Phase 6(c): Stochastic Quantitative Analysis. This phase consists in simulating different scenarios by modifying the initial marking with stochastic input event flows, thus obtaining the quantitative results for those scenarios. CPN Tools provides a simulator engine, which allows us to automatically replicate simulations of a scenario using its monitor capabilities. This is an important advantage of using CPN Tools: we can obtain relevant performance measures through simulation experiments, using the monitor features of CPN Tools. As previously mentioned, monitors are used to observe, inspect, or control simulations. In particular, we use data-collector monitors, which are used to extract numerical data from a PCPN. The numerical data obtained are then used to compute the statistic information.

To accomplish this objective, we use synthetic data that represent different scenarios. Thus, a new CPN Tools page was included in the PCPN model to produce the initial markings for different scenarios (Phase 6(a)), starting from a certain value of *CO* pollutant and increasing it by using a probability

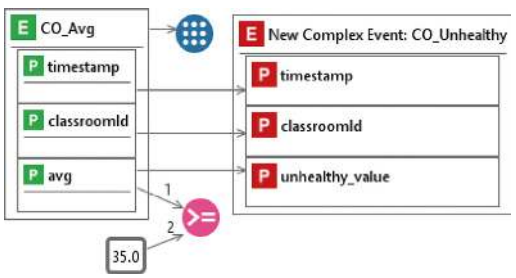


(a)

```
@Name('CO_Avg')
insert into CO_Avg
select a1.timestamp as timestamp,
  a1.classroomId as classroomId,
  avg(a1.value) as avg from
pattern [(every a1 = CO)].
  win:time(60 minutes)
group by a1.classroomId
```

(b)

Figure 7: CO_Avg event pattern. (a) CO Avg in MEdit4CEP-CPN. (b) CO Avg in EPL.



(a)

```
@Name('CO_Unhealthy')
insert into CO_Unhealthy
select a1.timestamp as timestamp,
  a1.classroomId as classroomId,
  a1.avg as unhealthy_value from
pattern [(every a1 = CO_Avg(a1.avg
  >= 35.0))]
```

(b)

Figure 8: CO_Unhealthy event pattern. (a) CO Unhealthy in MEdit4CEP-CPN. (b) CO Unhealthy in EPL.

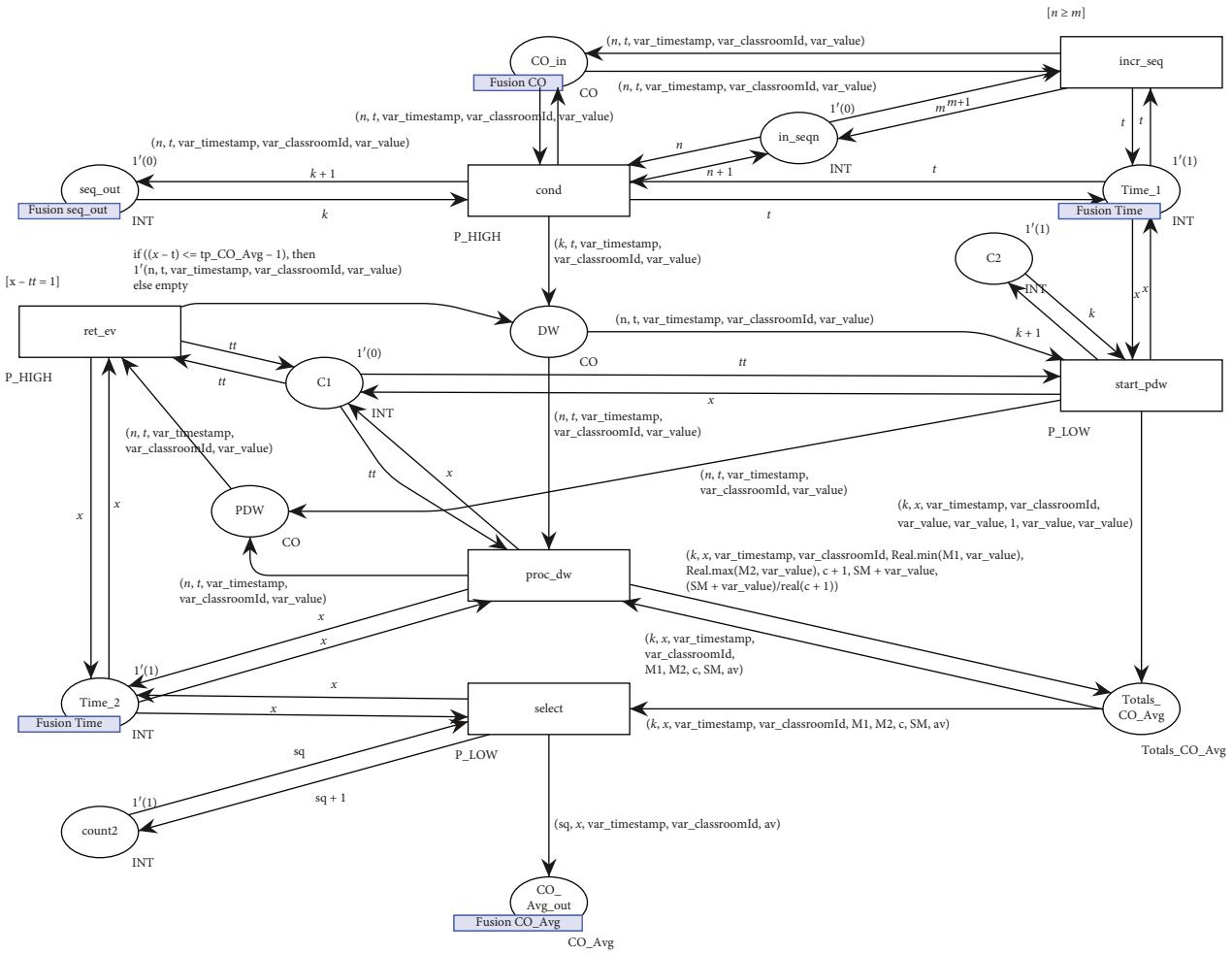


Figure 9: CO_Avg page.

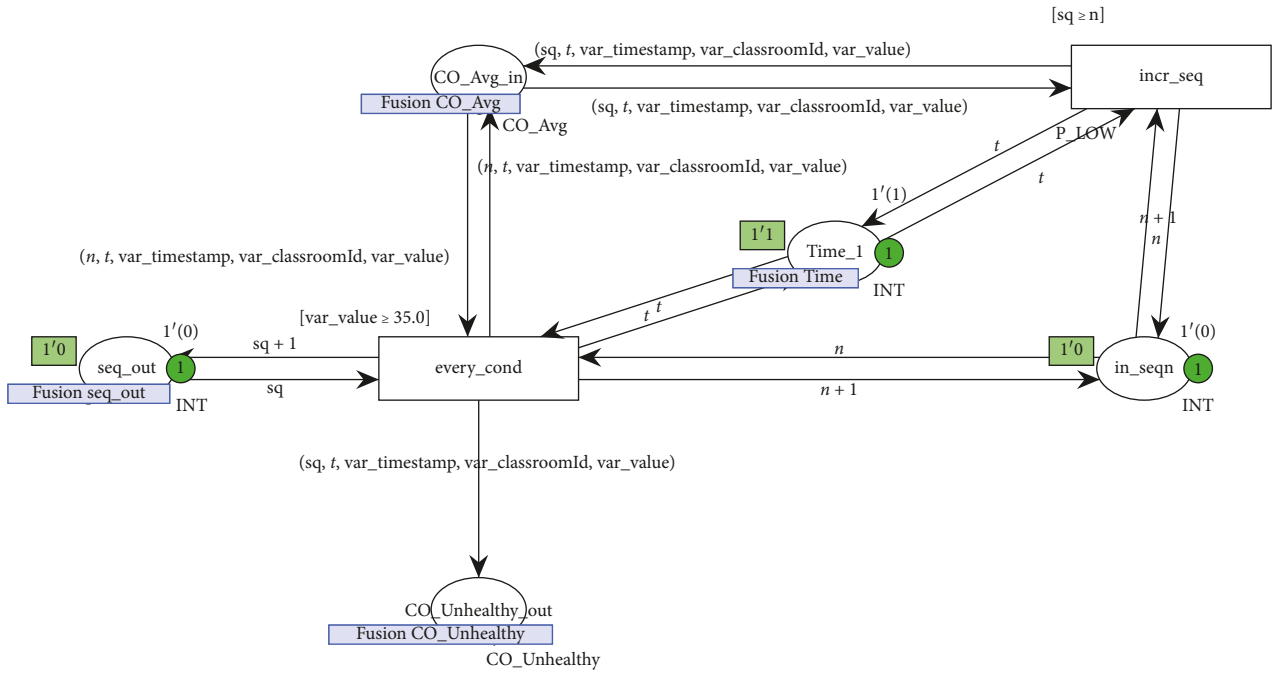


Figure 10: CO_Unhealthy page.

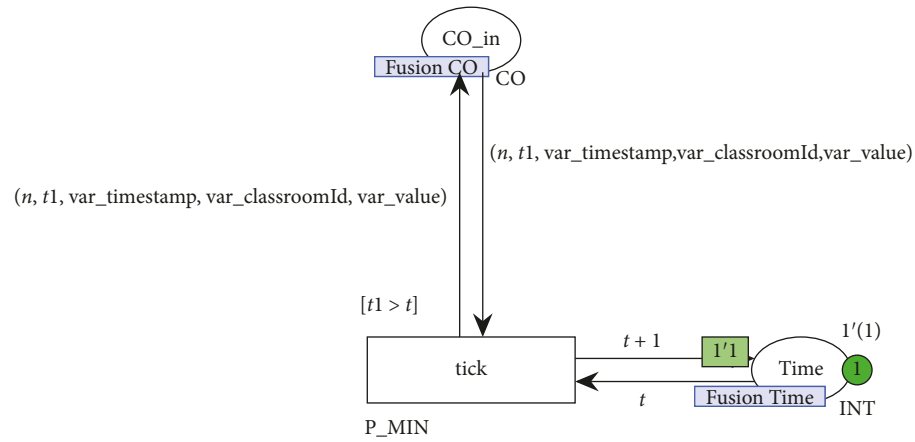


Figure 11: Events page.

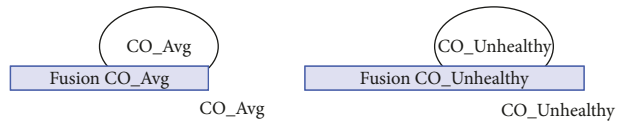


Figure 12: Complex events page.

distribution (see Figure 15). In this figure, transition *tinitial* fires 61 times in order to produce 61 events on place *CO_in* (we produce 61 events to allow the sliding time window to be processed for the first 60 events because we need the clock to reach the value 61). With each firing, the *CO* value (represented with the variable xx) on place *initial* is updated, by increasing it with a value obtained from a uniform distribution with arguments $f1$ and $f2$ (xx + uniform ($f1$, $f2$)). By changing the initial *CO* value, the parameters and/or the probability distribution function, we can easily generate different scenarios.

We now apply the monitor features of CPN Tools, which allow us to observe, inspect, control, or modify a simulation of a CPN. We consider two situations of interest in this work. The first checks whether the scenario reaches an unhealthy situation and the second the time of the first occurrence of this unhealthy situation. For this purpose, two monitors are specified, respectively. The first monitor (*reach_place_unhealthy*) is a place content break point monitor, and it will indicate us if there is at least one token in place CO_Unhealthy of Figure 14, and the second one (*time_first_unhealthy*) is a data-collector monitor, which

```
(1) (* Standard priorities *)
(2)    val P_MAX = 10;
(3)    val P_HIGH = 100;
(4)    val P_NORMAL = 1000;
(5)    val P_LOW = 10000;
(6)    val P_MIN = 20000;
(7) (* Standard declarations *)
(8)    colset INT = int;
(9)    colset STRING = string;
(10)   colset REAL = real;
(11)   colset BOOL = bool;
(12)   colset UNIT = unit;
(13)   colset TIME = time;
(14)   colset INTINF = intinf;
(15) (* Declarations for domain: SBS_CO *)
(16)   colset CO = product INT * INT * INT * STRING * REAL;
(17)   (* vars_for_event: CO *)
(18)      var var_timestamp: INT;
(19)      var var_classroomId: STRING;
(20)      var var_value: REAL;
(21)   var n, t, t1: INT;
(22) (* Declarations for complex events domain: SBS_CO *)
(23)   colset CO_Avg = product INT * INT * INT * STRING * REAL;
(24)   (* vars_for_complexevent: CO_Avg *)
(25)      var var_avg: REAL;
(26)   colset CO_Unhealthy = product INT * INT * INT * STRING * REAL;
(27) (* vars_for_complexevent: CO_Unhealthy *)
(28)   var var_unhealthy_value: REAL;
(29) (* Total color set for: CO_Avg *)
(30)   colset Totals_CO_Avg = product INT * INT * STRING * REAL * REAL * INT * REAL * REAL;
(31) (* Pattern auxiliary variables *)
(32)   var m, sq, k, tt, x, m1, m2, c, sm: INT;
(33)   var M1, M2, SM, av: REAL;
(34) (* Sliding time interval declarations: *)
(35)   val tP_CO_Avg = 12;
(36) (* Pattern auxiliary variables *)
(37) (* Declarations initial marking *)
(38)   colset INT3 = product INT * INT * INT;
(39)   var xx:REAL;
(40)   val co0 = 3.0;
(41)   val f1 = 0.5;
(42)   val f2 = 0.5;
```

Listing 1: CPN Tools declaration.

indicates us the time at which the place was reached for the first time. Both monitors are shown in Figure 16. In the first monitor, we only indicate the name of the place that will stop the execution if it becomes marked (*CO_unhealthy*), while in the second monitor, we need to write a predicate indicating the stop condition and an *Observer* function to obtain the time at which the execution stopped.

Once, the initial configuration is established, we can use the CPN Tools simulator engine to automatically replicate the experiment n times so as to obtain performance results. For instance, we can replicate it 100 times by using the following code:

$$\text{CPN}'\text{Replications.nreplications 100.} \quad (3)$$

Next, we introduce a specific case study, shown in Table 1, with 9 different scenarios. The objective of this study is to examine the probability of reaching an unhealthy condition before the 300 minutes. In these scenarios, the initial value for *CO* in all the simulations is (3.0 mg/m^3) and 100 replications have been used to obtain the results. As an illustration, in the third row corresponding to the third scenario, each time the *CO* value is updated in the initial place (see Figure 15), its value is increased by an arbitrary increment between 0.5 and 0.65. Therefore, the parameters for the uniform distribution function uniform ($f1$, $f2$) are established to $f1 = 0.5$ and $f2 = 0.65$. In this experiment, only 4% of the simulations reached an unhealthy situation. Notice that when the increment arguments are $f1 = 0.5$ and $f2 = 0.72$ (last row in the table), we always obtain an unhealthy situation (100%).

Other distribution functions can be considered as well. For instance, we can consider a normal distribution normal ($f1$, $f2$)

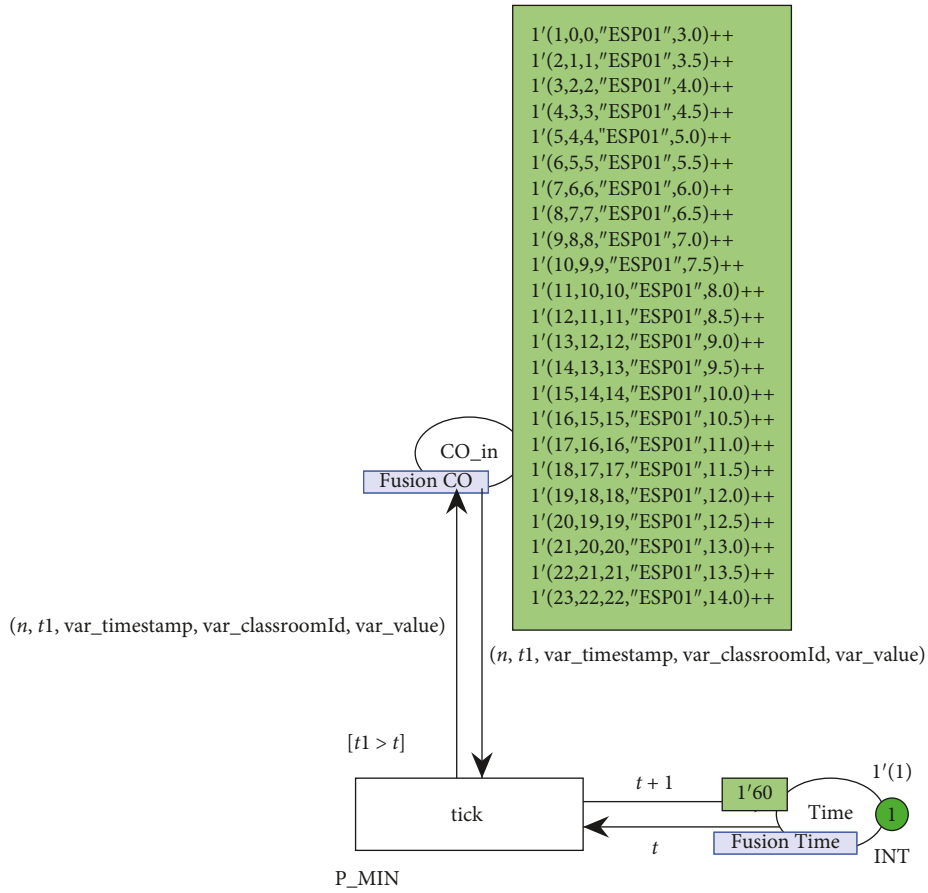


Figure 13: Initial marking in the event page.

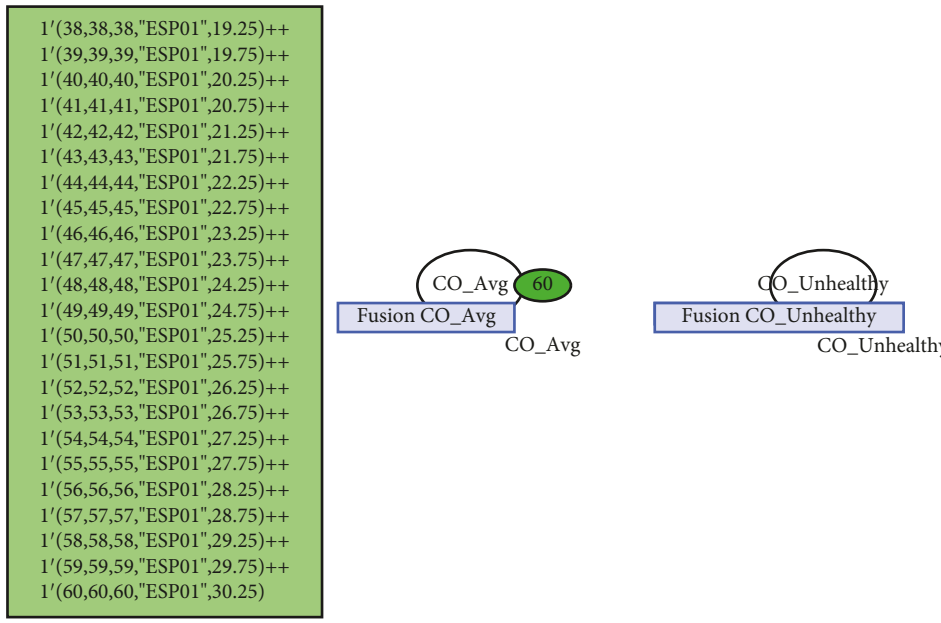


Figure 14: Final marking page.

to produce the events in the initial marking, as well as considering the initial value of CO of 3.0 mg/m^3. In particular, in the scenarios obtained for the normal distribution with parameters $f1 = 1.5$ and $f2 = 0.25, 0.5$, and 0.75, an unhealthy situation is always reached before 300 minutes. However, the time to reach this situation varies depending on the value of

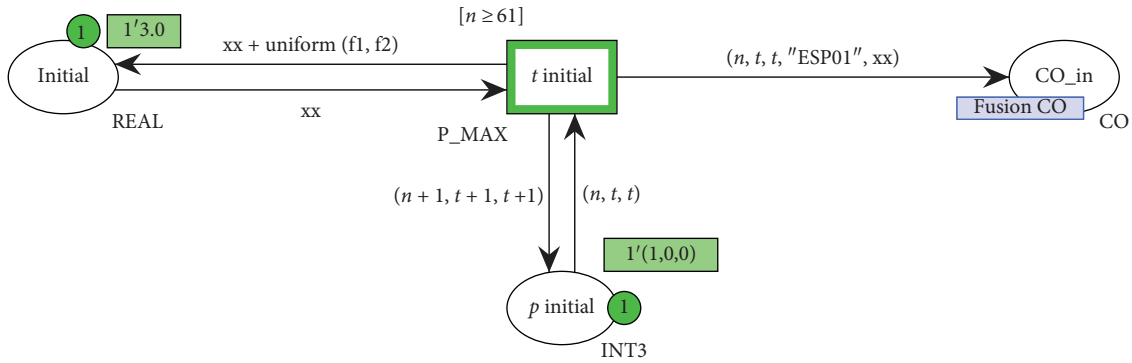


Figure 15: Initial marking page.

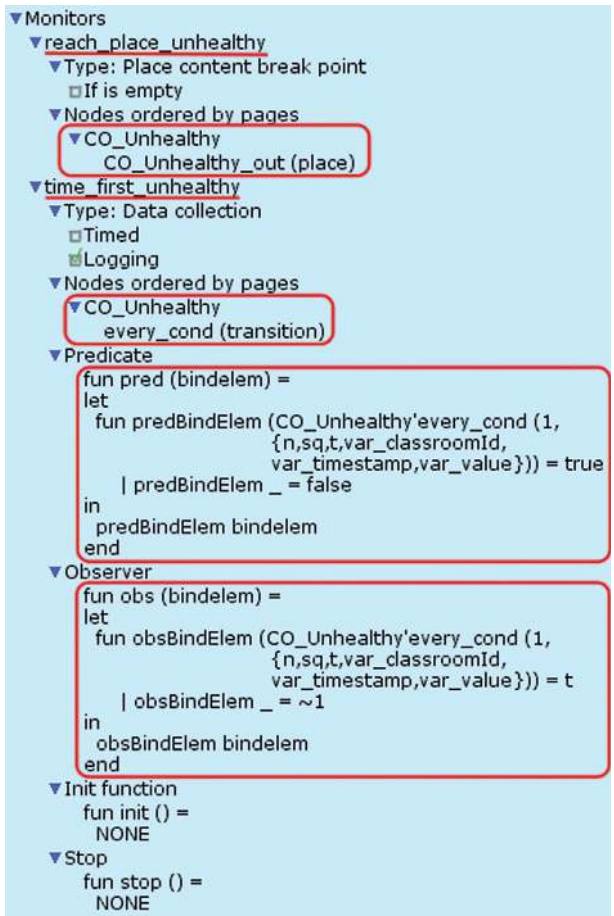


Figure 16: Monitors used in the scenario of Figure 13.

these parameters. Therefore, the objective now is to determine the time at which the unhealthy condition is reached. To study this circumstance, we proceed as in the previous case, by performing 100 replications with the values for parameters $f1$ and $f2$ indicated above. Figure 17 shows the results obtained using a *box plot* (the central mark in red indicates the median, and the bottom and top edges of the blue box indicate the first and third quartiles, respectively). The whiskers (discontinuous lines drawn in black from the box) extend to the most extreme data points not considered outliers, where the outliers are plotted using the "+" symbol in red).

Table 1: Results obtained taking different values to increase the *CO* level.

Scenario	Initial	Increment	Probability (%)
1	3	0.5 (fixed)	0
2	3	Uniform (0.5, 0.6)	0
3	3	Uniform (0.5, 0.65)	4
4	3	Uniform (0.5, 0.67)	41
5	3	Uniform (0.5, 0.68)	67
6	3	Uniform (0.5, 0.69)	88
7	3	Uniform (0.5, 0.7)	91
8	3	Uniform (0.5, 0.71)	99
9	3	Uniform (0.5, 0.72)	100

Notice that we have chosen a synthetic scenario to illustrate the applicability of our methodology, i.e., we start with a specific value for the *CO* value and we use different distributions to produce the event information for the next 300 minutes. Other arrangements can be considered, by modifying the initial value and/or the distribution used to change the values in credible values throughout the time. Obviously, the interest of using CPNs and CPN Tools lies in the possibility of using the automatic simulator engine and the monitor capabilities to make performance studies and obtain predictive results of the system behavior.

Finally, the model could be enriched by including the actions that should be taken to deal with the situations detected. As an example, for a detected unhealthy situation (*CO* average is greater than 35 mg/m^3 for 1 hour), we could consider as possible actions to start the air conditioner, open some windows, start some fans, etc. These actions could actually be included in the PCPN model as transitions that would be executed in the case that these conditions were satisfied. Furthermore, we could also use real data obtained from sensors in order to simulate a scenario with this pollution information. Thus, with the results obtained, we could predict unhealthy situations and then start the fans or the air conditioners before they occur.

5. Related Work

Over recent years, some collective computational intelligence technologies and tools have emerged in response to the demands for analyzing big data. Among them, the CEP technology is essential for analyzing the complexity of

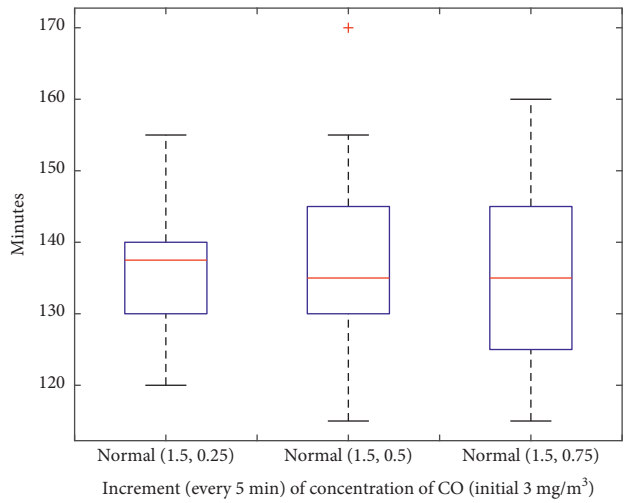


Figure 17: Box plot obtained for different scenarios.

multicriteria that could generate an alert [21], extracting meaningful events. However, the implementation of event pattern conditions can become a handicap for users who are experts on the domain but not in the involved technology. Moreover, conducting the generation of data for semantically validating those patterns is a cumbersome task. To solve these problems, on one hand, some works have proposed the use of Petri nets for conducting the semantic validation of event patterns, while others use Model-Driven Engineering (MDE) techniques for making the CEP technology closer to any user. We propose the combination of both approaches.

Regarding works using the Petri Net formalism, Offel et al. [22] show that formal methods as Petri nets can help in the design and implementation of CEP systems which are underdeveloped, but they are in the process of developing tool support for the envisioned verification of CEP systems. Weidlich et al. [23] used PCPNs with time in order to define a model of event-processing networks. The Event-Processing Network (EPN) architecture is presented, and a general translation of this concept, an example implemented in the ETALIS framework [24], is also presented.

Ahmad et al. [25] describe a methodology modeling CEP using Timed Net Condition Event System (TNCES) [26]. An application to a Manufacturing Line is also presented as an example. NCES is a Petri Net derived formalism based on Condition Event Systems to model discrete event dynamic systems. NCES is extended with time in TNCES, which is based on timed-arc Petri nets [27–29]. Thus, the main difference to our work, other than the Petri Nets formalism used, is that we integrate the PCPN translation into the MEdit4CEP tool, so as to automatically obtain the PCPNs from the event pattern graphical specification created by using this tool.

Other authors like Metzger et al. [30] analyze the CEP systems under the verification perspective applying model checkers. As an example, the authors perform incremental verifications using Petri Nets as models for the Tapaal tool, a bounded model checker. The approach used in this work to deal with the state explosion problem is to gradually increase the size of the model, which is a different approach to analyze CEP systems instead of using the quantitative analysis. Cugola and Margara [31] have defined the TESLA language, which is a complex event specification language, based on a metric temporal logic. TESLA is a highly expressive and flexible language in a rigorous framework, offering content and temporal filters, negations, timers, aggregates, and fully customizable policies for event selection and consumption. Ericsson et al. [32] have defined a prototype tool REX, with support for specifying both CEP systems and correctness properties in a high-level graphical language. CEP applications are then transformed into timed automata, and the UPPAAL tool [33] is used for automatic verification. Agrawal et al. [34, 35] have also defined a timed automata formalization of complex event systems. They present the Sase+pattern language, which defines a precise semantics in terms of timed automata with similar results to the work introduced in TESLA. Cugola and Margara have also proposed CAVE tool [36] with the purpose of assisting domain experts in the definition of a set of reliable rules for a CEP application. In particular, this tool analyzes the behavior of a CEP application transforming the property checking rules into basic constraints solving problems. Additionally, they have also presented a survey [37] of the existing Information Flow Processing (IFP) systems, including CEP systems, activa databases, etc. They show the different approaches and mechanisms adopted in these IFP systems to deal with the event flow processing.

Rewriting logic and the Maude language [38] have been also used to specify and analyze CEP systems. Burgueño et al. [39] propose a framework for the specification of CEP applications, allowing developers to formally analyze and prove properties of their CEP programs. An encoding of CEP concepts and mechanisms to Maude is provided, and several analysis are presented, both covering the static properties of the CEP patterns and the statistical simulation of such systems. Garcí-a-López et al. [40] have implemented the CEPA tool for the transformation of CEP programs to Abstract Syntax Trees (AST) capturing the pattern dependencies, with the goal to check and correct two particular

Table 2: Modeling/analysis approaches based on formal methods.

Approach	Objective	Underlying formalism and tool	Modeling	Model validation	Transformation
MEdit4CEP-CPN [6]	Semantic and quantitative analysis graphically supported by CPN Tools	Colored Petri nets & CPN Tools	Graphical	Syntactic analysis of graphical models	Automatic to EPL and PCPN
CEP in Maude [39]	General formal analysis script supported by Maude	Rewriting logic and Maude	Textual	Syntactic analysis	Manual
CEPA [40]	Rule acyclicity and race condition checking	Graph theory	Textual	Syntactic analysis of textual models	Automatic to AST
REX [32]	Property verification	Timed automata	Graphical	Using UPPAAL	Automatic to timed automata
CAVE [36]	Property verification and performance analysis	Ad hoc CEP language	Textual	Constraint solver	Automatic to configurations

properties of CEP systems: rule acyclicity and rule race conditions.

Concerning the quantitative analysis of complex events, Tendick et al. [41] focus on the use of statistical methods for the CEP technology for making decisions in real time, providing additionally a comparison of computing techniques in widespread use for real-time data. Other approaches like Rajsiri et al.'s work [42] and the one presented here focus on studios via simulations. Rajsiri et al. present a business process editor and simulator developed on the basis of an event-driven business process modeling approach using the BPMN 2.0 formalism. This work dealt with the problem of the business processes simulation taking an even-driven perspective into account to observe the system behavior; however, users cannot automatically replicate scenarios. In addition, there are also several tools for performance evaluation of CEP applications by simulations, for instance, CEPSim [43] is a simulator for CEP and Stream Processing (SP) systems in cloud environments which allows us to analyze the performance and scalability of user-defined queries and to evaluate the effects of various query processing strategies. Mendes et al. [44] have developed FINCoS, which is a set of benchmarking tools for load generation and performance measuring of event-processing systems, so as to make performance evaluation on CEP platforms independently on their structural differences or the workload employed. Along the same lines, Li and Berry [45] have also developed a benchmark of complex event-processing systems focusing on complex event-processing functional behaviours: filtering, transformation, and event pattern detection. They also show the factors that influence performance measurements.

As a summary, Table 2 shows a comparison of our proposal (MEdit4CEP-CPN) with the most representative mentioned modeling/analysis works based on formal methods.

It also deserves special mention the CEP engine proposed by Cugola and Margara [46], T-REX, based on the TESLA language that combines expressiveness and efficiency. T-REX middleware provides an efficient event detection algorithm based on automata to interpret TESLA rules. This work could be used in conjunction to our proposal through the inclusion of additional transformations to generate TESLA rules.

Regarding to works considering metamodels of PNs, Gomez et al. [47] proposed a metamodel for PNs in the domain of *biological data processing*. The models conforming to this metamodel are then transformed into the XML code executable by CPN Tools. This work shows some limitations with respect to our proposal. Among them, PN modeling is close to CPN Tools concepts, and a tree model editor is used to produce the models. In addition, a CPN model can only have one page, and priorities are not considered. Additionally, Westergaard et al. [48] implemented Access/CPN, a framework providing CPN Tools with two interfaces. One is written in Standard Markup Language, which is useful for analysis methods. The other interface is written in Java and provides an object-oriented representation of CPN models, whose object model (metamodel) is implemented by using Eclipse Modeling Framework (EMF). However, the latest version released is not actually up-to-date and, although the latest version available from Subversion (https://svn.win.tue.nl/repos/cpntools/AccessCPN/trunk/) has better support for 4.0 features of CPN Tools, it is still not complete, as stated by Westergaard. In addition, Petri Net modeling is addressed by using a tree model editor (not a graphical one with nodes and links), as in the work by Gomez et al. [47].

6. Conclusions and Future Work

In this paper, we have illustrated the use of the MEdit4CEP-CPN approach for the complex event analysis through a case study based on the sick building syndrome. The event patterns have been graphically modeled with MEdit4CEP-CPN and then automatically transformed into both EPL and CPN code. Additionally, CPN Tools have been used to make quantitative analysis of events produced for this case study. Given the flexibility provided by MEdit4CEP-CPN, this analysis could be applied to other cutting-edge real-world case studies, such as eHealth [49], robotic [50] and mobile edge, and cloud computing applications [51].

As shown in the related work, there are many works using CPNs to model CEP-based languages, but to the best of our knowledge, they do not provide end users with an all-in-one graphical tool with the following goals: (1) modeling CEP domains and event patterns in a user-friendly way by dragging and dropping elements on a canvas, (2) validating the pattern syntax, (3) automatically transforming the graphical patterns

into a CPN model, (4) automatically transforming the CPN model to the XML code executable by CPN Tools and validating the pattern semantics, (5) automatically generating the Esper EPL code and deploying it in a particular event-based system, and (6) providing a quantitative analysis of complex events through the CPN Tools executable model automatically generated by the tool. Let us observe that the MEdit4CEP-CPN model-driven approach presented in this paper provides support for all of these functionalities.

As future work, we plan to add additional features and functionalities to MEdit4CEP-CPN, such as further EPL operators and new transformation techniques. Since the use of CPN Tools requires some knowledge from users in order to conduct the quantitative analysis, at least for modifying the initial marking of the produced CPN model and to execute the simulations to obtain the results, we intend to alleviate this problem by enriching our graphical model for event pattern design. This will make it possible to set the initial conditions (event flow) at design time and adding the option to automatically execute the produced CPN. The obtained output would then be transformed into the corresponding complex events in the output flow.

Data Availability

The obtained PCPN model and simulation data used to support the findings of this study have been deposited in the Mendeley repository (DOI: 10.17632/kfrkyzdxnv.1).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work has been partially supported by the Spanish MINECO/FEDER projects TIN2015-65845-C3-2-R, TIN2015-65845-C3-3-R, and TIN2016-81978-REDT. Boubeta-Puig would like to thank the Real-Time and Concurrent Systems Research Group for their hospitality when visiting them at the University of Castilla-La Mancha, Spain, where part of this work was developed.

References

[1] D. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*, John Wiley & Sons, New Jersey, NJ, USA, 2012.

[2] F. Bry, M. Eckert, O. Etzion, J. Riecke, and P. Adrian, "Event processing languages. Tutorial in DEBS 2009," in *Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems*, Nashville, TN, USA, July 2009.

[3] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: a model-driven solution for real-time decision making in SOA 2.0," *Knowledge-Based Systems*, vol. 89, pp. 97–112, 2015.

[4] EsperTech, "Esper—complex event processing," February 2019, http://www.espertech.com/esper/.

[5] W. M. P. van der Aalst and C. Stahl, *Modeling Business Processes—A Petri Net-Oriented Approach. Cooperative Information Systems Series*, MIT Press, Cambridge, MA, USA, 2011.

[6] J. Boubeta-Puig, G. Díaz, H. Macià, V. Valero, and G. Ortiz, "MEdit4CEP-CPN: an approach for complex event processing modeling by prioritized colored Petri nets," *Information Systems*, vol. 81, pp. 267–289, 2019.

[7] H. Macia, V. Valero, G. Diaz, J. Boubeta-Puig, and G. Ortiz, "Complex event processing modeling by prioritized colored Petri nets," *IEEE Access*, vol. 4, pp. 7425–7439, 2016.

[8] M. Westergaard and H. M. W. Verbeek, "CPN tools homepage," February 2019, http://www.cpntools.org/.

[9] A. Adi, D. Botzer, N. Gil, and S. Guy, "Complex event processing for financial services," in *Proceedings of the 2006 IEEE Services Computing Workshops*, pp. 7–12, IEEE, Chicago, IL, USA, September 2006.

[10] A. Buchmann, H. C. Pfohl, S. Appel et al., "Event-driven services: integrating production, logistics and transportation," in *Service-Oriented Computing*, E. Michael Maximilien, G. Rossi, S.-T. Yuan, H. Ludwig, and M. Fantinato, Eds., vol. 6568, pp. 237–241, Springer, Berlin, Germany, 2011.

[11] R. Gad, M. Kappes, J. Boubeta-Puig, and I. Medina-Bulo, "Employing the CEP paradigm for network analysis and surveillance," in *Proceedings of the Ninth Advanced International Conference on Telecommunications*, pp. 204–210, IARIA, Rome, Italy, June 2013.

[12] A. Garcia-de-Prado, G. Ortiz, and J. Boubeta-Puig, "COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things," *Expert Systems with Applications*, vol. 85, pp. 231–248, 2017.

[13] A. Garcia-de Prado, G. O. Ortiz, J. Boubeta-Puig, and D. Corral-Plaza, "Air4People: a smart air quality monitoring and context-aware notification system," *Journal of Universal Computer Science*, vol. 24, no. 7, pp. 846–863, 2018.

[14] W. Wang, H. Yang, Y. Zhang, and J. Xu, "IoT-enabled real-time energy efficiency optimisation method for energy-intensive manufacturing enterprises," *International Journal of Computer Integrated Manufacturing*, vol. 31, no. 4-5, pp. 362–379, 2018.

[15] M. E. Cambronero, H. Macia, V. Valero, and L. Orozco-Barbosa, "Modeling and analysis of the 1-wire communication protocol using timed colored Petri nets," *IEEE Access*, vol. 6, pp. 27356–27372, 2018.

[16] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, Springer Publishing Company, Berlin, Germany, 1st edition, 2009.

[17] Eclipse Foundation, "Epsilon," June 2019, https://www.eclipse.org/epsilon/.

[18] D. S. Kolovos, A. García-Domínguez, L. M. Rose, and R. F. Paige, "Eugenia: towards disciplined and automated development of GMF-based graphical model editors," *Software & Systems Modeling*, vol. 16, no. 1, pp. 229–255, 2017.

[19] World Health Organization, *Regional Office for Europe*, WHO, Geneva, Switzerland, February 2019, http://www.euro.who.int/en/home.

[20] World Health Organization, *Regional Office for Europe*, WHO Guidelines for Indoor Air Quality: Selected Pollutants, Geneva, Switzerland, February 2019, https://apps.who.int/iris/bitstream/handle/10665/260127/9789289002134-eng.pdf?sequence=1&isAllowed=y.

[21] N. Bessis and F. Xhafa, *Next Generation Data Technologies for Collective Computational Intelligence*, Springer, Berlin, Germany, 2011.

[22] M. Offel, H. van der Aa, and M. Weidlich, "Towards net-based formal methods for complex event processing," in *Proceedings*

of the Conference "Lernen, Wissen, Daten, Analysen", LWDA 2018, pp. 281–284, Mannheim, Germany, August, 2018.

[23] M. Weidlich, M. Jan, and A. Gal, "Net-based analysis of event processing networks—the fast flower delivery case," in *Application and Theory of Petri Nets and Concurrency, LNCS*, J. M. Colom and J. Desel, Eds., vol. 7927, pp. 270–290, Springer, Berlin, Germany, 2013.

[24] D. Anicic and P. Fodor, "Event-driven transaction logic inference system," February 2019, https://code.google.com/archive/p/etalis/.

[25] W. Ahmad, A. Lobov, L. Jose, and M. Lastra, "Formal modelling of complex event processing: a generic algorithm and its application to a manufacturing line," in *Proceedings of the IEEE 10th International Conference on Industrial Informatics, INDIN*, pp. 380–385, Beijing, China, July 2012.

[26] M. Rausch and H. M. Hanisch, "Net condition/event systems with multiple condition outputs," in *Proceedings of the Emerging Technologies and Factory Automation, ETFA'95*, pp. 592–600, Paris, France, October 1995.

[27] D. de Frutos, V. Valero, and O. Marroquín, "Decidability of properties of timed-arc Petri nets," in *Application and Theory of Petri Nets, LNCS*, M. Nielsen and D. Simpson, Eds., vol. 1825, pp. 187–206, Springer, Berlin, Germany, 2000.

[28] H.-M. Hanisch, "Analysis of place/transition nets with timed-arcs and its application to batch process control," in *Application and Theory of Petri Nets, LNCS*, M. Ajmone Marsan, Ed., vol. 691, pp. 282–299, Springer, Berlin, Germany, 1993.

[29] V. Valero, D. de-Frutos, and F. Cuartero, "On non-decidability of reachability for timed-arc Petri nets. Petri nets and performance models," in *Proceedings 8th International Workshop on Petri Nets and Performance Models*, pp. 188–197, IEEE Computer Society Press, Zaragoza, Spain, September 1999.

[30] A. Metzger, C. Reinartz, and K. Pohl, "Incremental verification of complex event processing applications for system monitoring," in *Proceedings of the 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 293–297, Prague, Czech Republic, August 2018.

[31] G. Cugola and A. Margara, "TESLA: a formally defined event specification language," in *Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems, DEBS '10*, pp. 50–61, ACM, Cambridge, UK, July 2010.

[32] A. M. Ericsson, P. Pettersson, M. Berndtsson, and M. Seiriö, "Seamless formal verification of complex event processing applications," in *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, DEBS*, pp. 50–61, Toronto, ON, Canada, June 2007.

[33] K. G. Larsen, P. Pettersson, and Y. Wang, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[34] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 147–160, ACM, Vancouver, Canada, June 2008.

[35] D. Gyllstrom, J. Agrawal, Y. Diao, and N. Immerman, "On supporting kleene closure over event streams," in *Proceedings of the 24th International Conference on Data Engineering (ICDE)*, pp. 1391–1393, Cancún, MX, USA, April 2008.

[36] G. Cugola, A. Margara, M. Pezzè, and M. Pradella, "Efficient analysis of event processing applications," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15*, pp. 10–21, Oslo, Norway, June-July 2015.

[37] G. Cugola and A. Margara, "Processing flows of information: from data stream to complex event processing," *ACM Computing Surveys*, vol. 44, no. 3, pp. 1–62, 2012.

[38] M. Clavel, F. Durán, S. Eker et al., "All about Maude—a high-performance logical framework, how to specify, program and verify systems in rewriting logic," in *Lecture Notes in Computer Science*, Vol. 4350, Springer, Berlin, Germany, 2007.

[39] L. Burgueno, J. Boubeta-Puig, and A. Vallecillo, "Formalizing complex event processing systems in Maude," *IEEE Access*, vol. 6, pp. 23222–23241, 2018.

[40] A. García-López, L. Burgueño, and A. Vallecillo, "Static analysis of complex event processing programs," in *Proceedings of the MODELS 2018 Workshops*, vol. 14, pp. 498–502, Copenhagen, Denmark, October 2018.

[41] P. H. Tendick, L. Denby, and W.-H. Ju, "Statistical methods for complex event processing and real time decision making," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 8, no. 1, pp. 5–26, 2016.

[42] V. Rajsiri, N. Fleury, C. Graham, and J.-P. Lorré, "Event-based business process editor and simulator," in *Business Process Management Workshops—BPM 2010 International Workshops and Education Track*, M. zur Muehlen and J. Su, Eds., Springer, Berlin, Germany, 2011.

[43] W. A. Higashino, M. A. M. Capretz, and L. F. Bittencourt, "CEPSim: modelling and simulation of complex event processing systems in cloud environments," *Future Generation Computer Systems*, vol. 65, pp. 122–139, 2016.

[44] M. R. N. Mendes, B. Pedro, and P. Marques, "FINCoS: benchmark tools for event processing systems," in *Proceedings of the ACM/SPEC on International conference on Performance Engineering—ICPE'13*, pp. 431-432, Prague, Czech Republic, August 2013.

[45] C. Li and R. Berry, "CEPBen: a benchmark for complex event processing systems," in *Performance Characterization and Benchmarking*, R. Nambiar and M. Poess, Eds., pp. 125–142, Springer International Publishing, Cham, Switzerland, 2014.

[46] G. Cugola and A. Margara, "Complex event processing with T-REX," *Journal of Systems and Software*, vol. 85, no. 8, pp. 1709–1728, 2012.

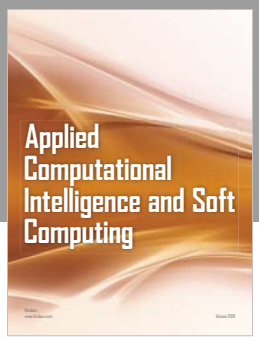
[47] A. Gomez, A. Boronat, J. A. Carsi, I. Ramos, C. Taubner, and S. Eckstein, "Biological data processing using model driven engineering," *IEEE Latin America Transactions*, vol. 6, no. 4, pp. 324–331, 2008.

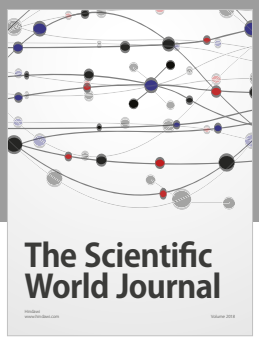
[48] M. Westergaard and L. Michael Kristensen, "The Access/CPN framework: a tool for interacting with the CPN Tools simulator," in *Applications and Theory of Petri Nets*, G. Franceschinis and K. Wolf, Eds., pp. 313–322, Springer, Berlin, Germany, 2009.

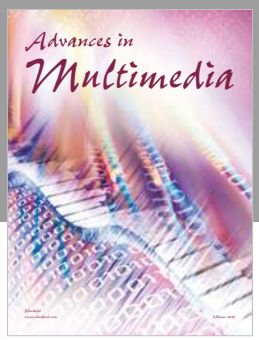
[49] A. Camacho, M. G. Merayo, and M. Núñez, "Collective intelligence and databases in eHealth: a survey," *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 2, pp. 1485–1496, 2017.

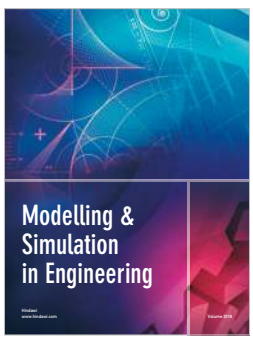
[50] A. Rutle, J. Backer, K. Foldøy, and R. T. Bye, "Commonlang: a DSL for defining robot tasks," in *Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS Co-Located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018)*, pp. 433–442, Copenhagen, Denmark, October 2018.

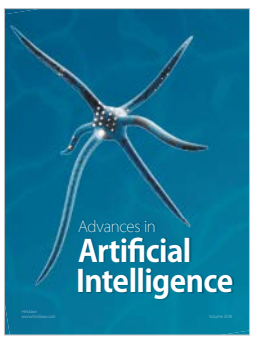
[51] G. Orsini, D. Bade, and W. Lamersdorf, "CloudAware: empowering context-aware self-adaptation for mobile applications," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 4, e3210 pages, 2018.

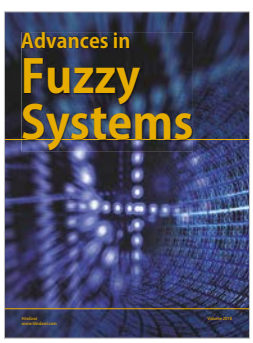


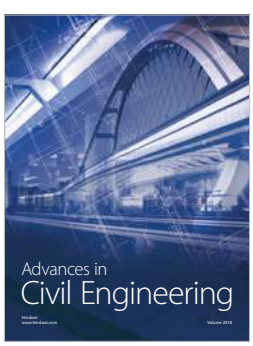


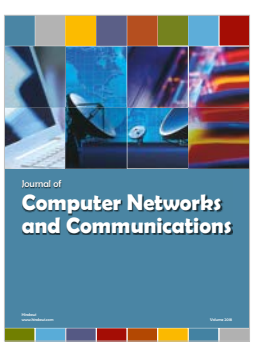


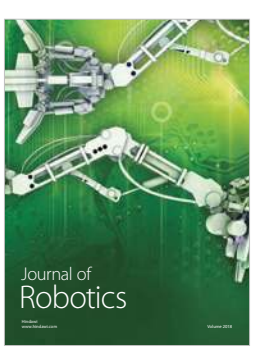


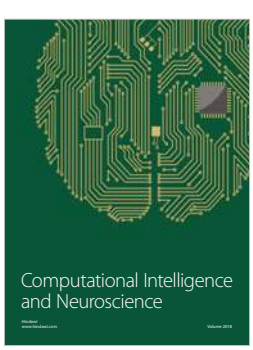












Hindawi

Submit your manuscripts at
www.hindawi.com