Petrinets: What are they ? How can help us?

Gabriel E. Valenzuela

Wazuh

November 2022

wazuh.

Agenda

Introduction

- About me and this talk
- A basic knowledge of systems

Petrinets 101

- Some particular structures
- Special type of arcs
- Guard & events
- Some considerations

Petrinets and C++

Q&A

About me and this talk

About me

- Computer engineer recently graduate from the Universidad Nacional de Córdoba (Argentina).
- Software engineer at Wazuh Inc.
- Volunteer in some conferences (Cpp on Sea, CppNow, CppCon).
- A very curious and pragmatic person, always trying to learn something new.

About me and this talk

About me

- Computer engineer recently graduate from the Universidad Nacional de Córdoba (Argentina).
- Software engineer at Wazuh Inc.
- Volunteer in some conferences (Cpp on Sea, CppNow, CppCon).
- A very curious and pragmatic person, always trying to learn something new.

This talk is about...

- Make a little review about model systems and tools.
- Explain what are the petri networks and it's benefits.
- Give an example and relationship between Petrinets & C++.

A basic knowledge of systems

- As businesses and systems have become more sophisticated and more complicate, there has been an increasing emphasis on speed in systems analysis and **design**.

A basic knowledge of systems

- As businesses and systems have become more sophisticated and more complicate, there has been an increasing emphasis on speed in systems analysis and **design**.
- Understanding systems development requires an **understanding** not only of each technique, tool, and method, but also of ***how*** these elements complement and support each other within an organizational setting.

A basic knowledge of systems

- As businesses and systems have become more sophisticated and more complicate, there has been an increasing emphasis on speed in systems analysis and **design**.
- Understanding systems development requires an **understanding** not only of each technique, tool, and method, but also of ***how*** these elements complement and support each other within an organizational setting.
- Whether the system type or the structure of the organization, most use the **systems development life cycle (SDLC)** as a common methodology; it features several phases that mark the progress of the systems analysis and design effort.

Systems design and its tools

The third phase in the SDLC is **design**. During design, analysts convert the description of the recommended alternative solution into logical and then physical system specifications.

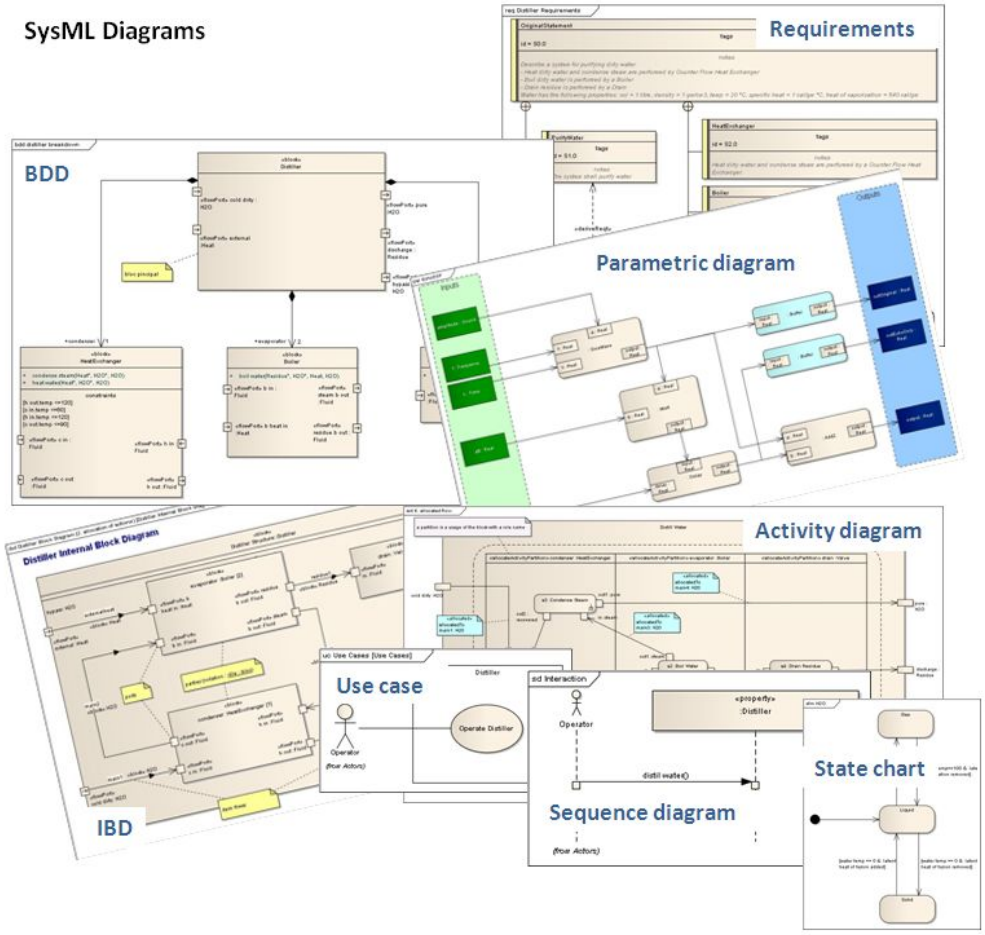
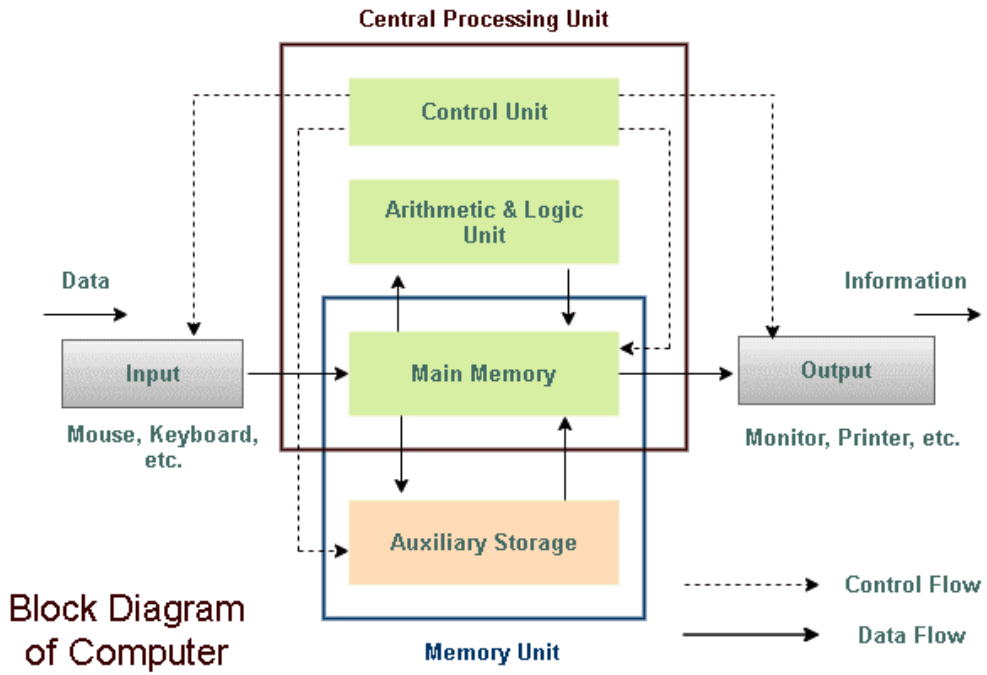


Figure: SysML

Figure: Block diagrams



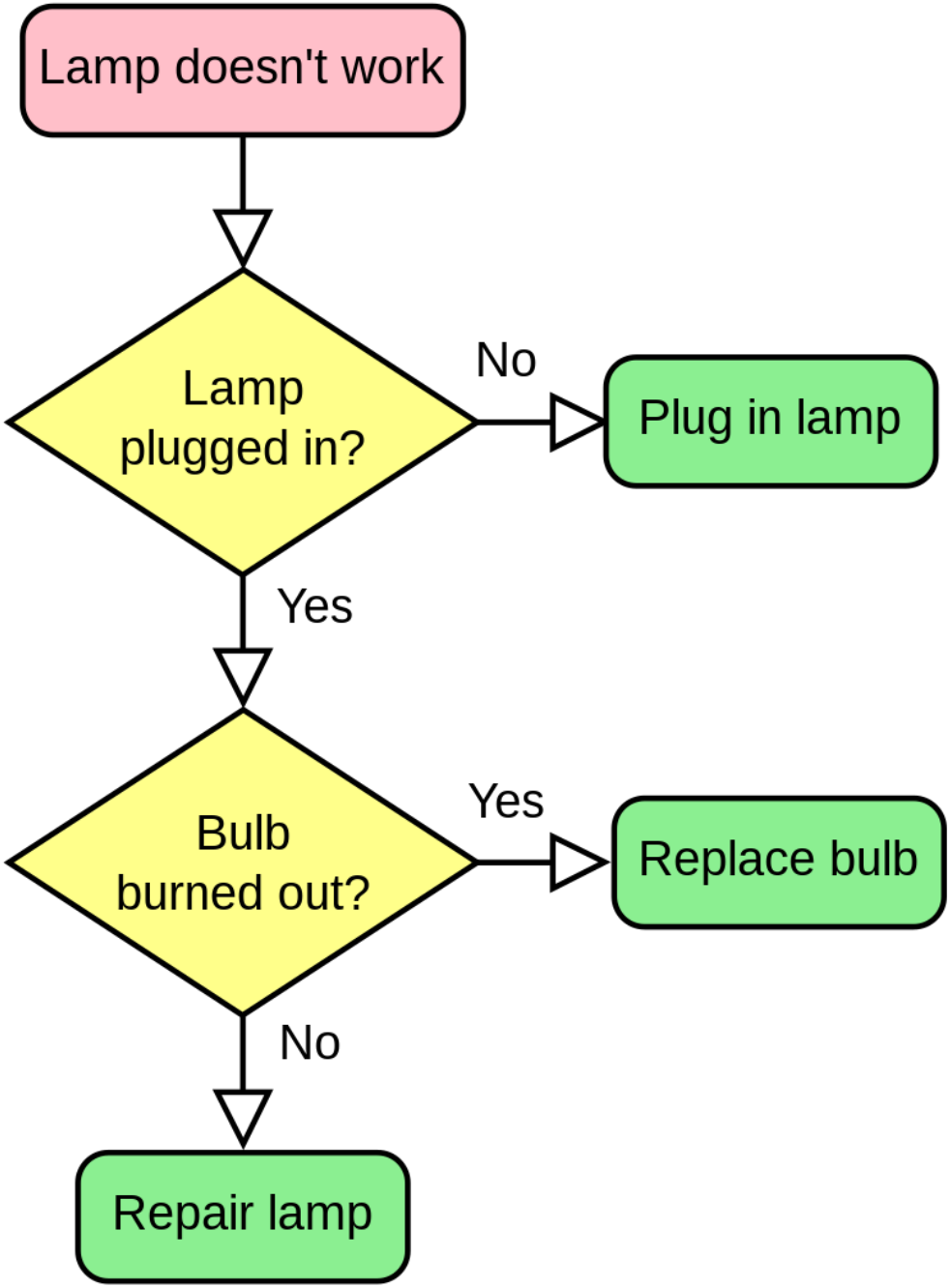


Figure: Flow diagrams

A brief history of Petrinets

- (PetriNets) PNs were introduced by **A.C. Petri** in *1962* to synchronize communicating automata, and were then extended to define a large set of models, with increasing complexity and capabilities.
- Some authors says Petri invented PNs in *1939* at the age of 13 for the purpose of modeling chemical processes. However, many refer to his Ph.D. thesis, defended in *1962*, as the starting point for Petri nets.

Figure: C. A. Petri (1926-2010)

Fundamental concepts

A Petri net (PN) it's a **bipartite** graph that has two types of nodes, namely **places** and **transitions**.

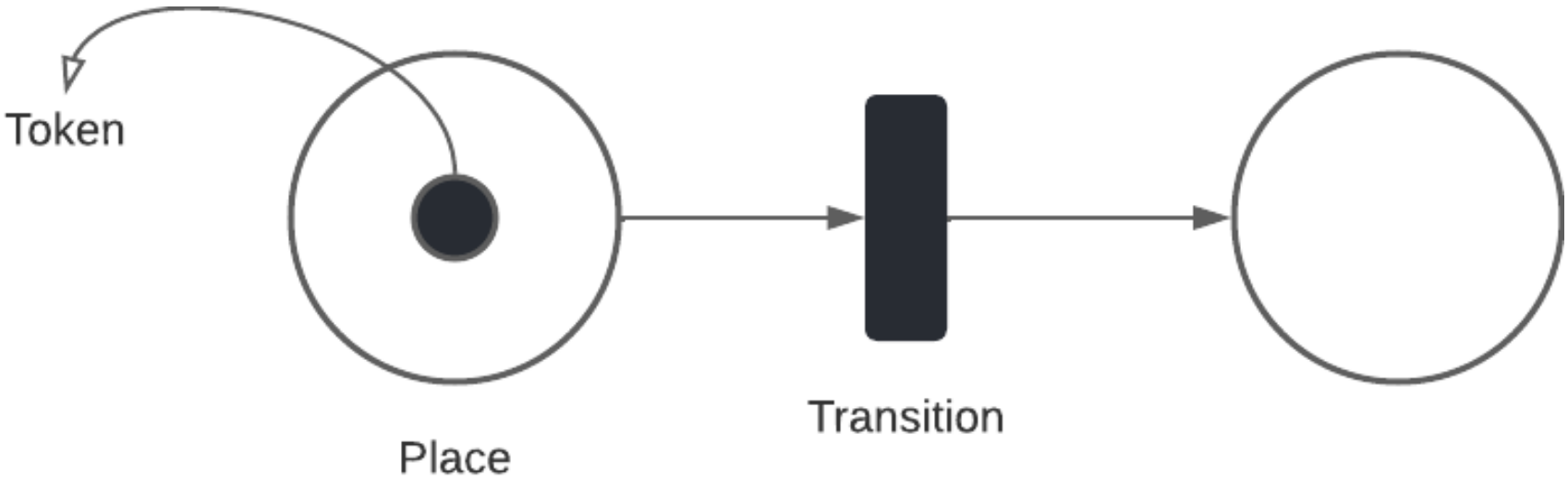


Figure: PN description

Fundamental concepts

A place is represented by a circle and a transition by a bar (certain authors represent a transition by a box). Places and transitions are connected by **arcs**. The number of places is finite and not zero. The number of transitions is also finite and not zero. An arc is directed and connects either a place to a transition or a transition to a place depending of the arc type.

Some particular structures

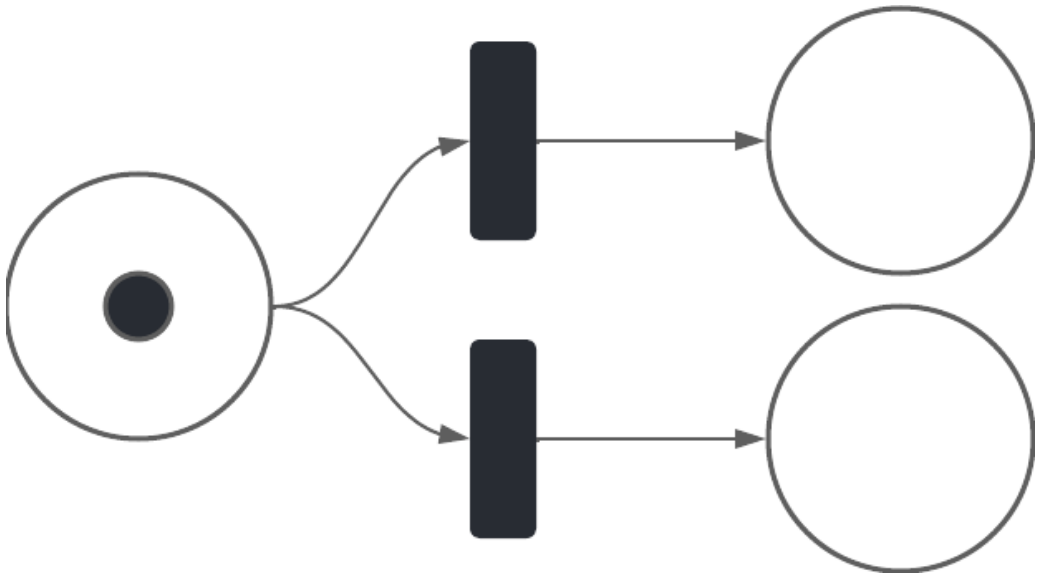


Figure: PN as state graph

Some particular structures

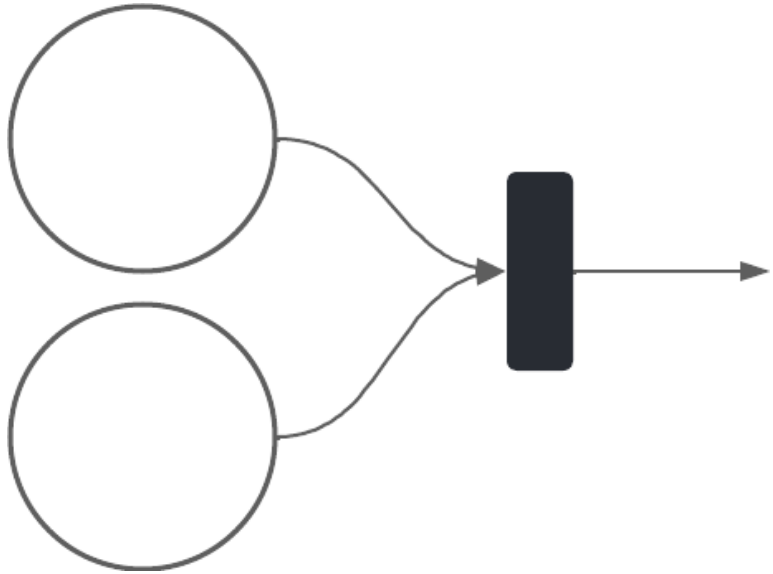


Figure: PN as event graph

General state equation

- It's important to introduce the PNs state equation in order to extend it with different types of arcs, events, guards and time. Through this equation, it is possible to obtain the next state of the system. This way is simpler than the graphic approach when analyzing the system's evolution.
- This equation is a biunivocal relationship between the graphical and mathematical expression.
- The PN state equation for a net with n places and m transitions, arcs with a weight greater or equal to one and an initial mark of M_0 , is:

State equation

$$M_{j+1} = M_j + I \cdot \sigma \tag{1}$$

General state equation

- Where: I is the incidence matrix, σ is the firing vector, with a size of $m \times 1$. M_0 is the initial marking vector, M_j is the current marking vector, and M_{j+1} is the next state's marking vector, all of them have a size of $n \times 1$.

General state equation

- Where: I is the incidence matrix, σ is the firing vector, with a size of $m \times 1$. M_0 is the initial marking vector, M_j is the current marking vector, and M_{j+1} is the next state's marking vector, all of them have a size of $n \times 1$.

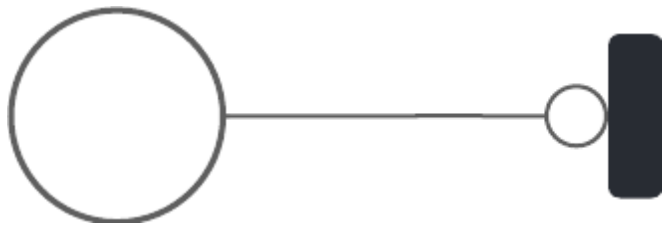
Incidence matrix

The incidence matrix, I, it's a matrix of dimensions $n \times m$, where its elements $I_{i,j}$ its define as:

$$I_{i,j} = w_i(t_i, p_j) - w_o(p_j, t_i)$$

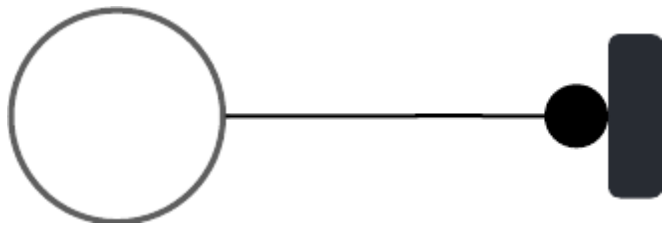
Where $w_i(t_i, p_j)$ it's the arc weight from transition-j to place-i and $w_o(p_j, t_i)$ it's the weight arc from place-i to transition-j.

Special type of arcs



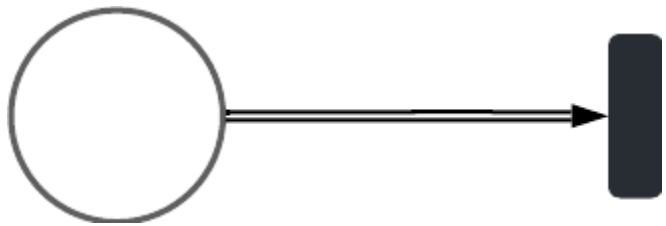
If the arc is an inhibitor arc, and the source place has at least one token, then the transition target is **not** enabled.

Special type of arcs



If the arc is a reader arc, and the source place hasn't any token, then the transition target is **not** enabled.Firing does not consume any tokens present.

Special type of arcs



If the arc is a reset arc, hen all tokens in source place are consumed. Firing sets the mark in source places to zero.

Guard & events

Guards

A guard is a logic variable associated to a transition that enhances its expression capability. Graphically, a guard is represented with a tag next to a transition. If and t_i is a transition and $g(t_i)$ is a guard associated to t_i , t_i can only be fired if it's enabled and if the guard's value is **true**.

Guard & events

Guards

A guard is a logic variable associated to a transition that enhances its expression capability. Graphically, a guard is represented with a tag next to a transition. If and t_i is a transition and $g(t_i)$ is a guard associated to t_i , t_i can only be fired if it's enabled and if the guard's value is **true**.

Events

An event is stored in a queue associated to a transition, enhancing the PNs expression capability. Graphically, the queue is represented with a tag next to a transition. The queue q_i is a counter, which increments each time a new event comes in, and decrements when the associated transition is fired. For the transition to be enabled, it is required for the queue to have at least **one event**.

Some considerations

- The **shot** selection policy consists in the selection of the next transition to be fired among all of the ones that are currently enabled. For PNs, including non-autonomous PNs and PNs with time, if this selection is random, the resulting system is non determin-istic. There are different solutions that provide deterministic behavior, such as including priorities, probabilities, inhibitor arcs, lector arcs, etc.In order to avoid conflicts among the transitions, we use the single server policy.This means that we calculate the new state (marking) considering only one shot of one transition. If multiple shots are required, a sequence of shots is performed. ,

Some considerations

- Once we define which is the next transition to be fired,the shot is performed with the original state equation, except for the reset arc, which removes every single token from the associated place.
- In order to mathematically represent the existence of the previously enumerated arcs, matrixes that indicate the connections between places and transitions are needed.These matrixes are similar to the I matrix. The matrixes's elements are *binary*, since the arc's weight is always one.So, for a transition to be enabled, the following conditions must be met:

Some considerations

- If it has an inhibitor arc, the place must **not have any** tokens

Some considerations

- If it has an inhibitor arc, the place must **not have any** tokens
- If it has a lector arc, the place must have **at least one token**.

Some considerations

- If it has an inhibitor arc, the place must **not have any** tokens
- If it has a lector arc, the place must have **at least one token**.
- If it has a guard, the guard value must be **true**.

Some considerations

- If it has an inhibitor arc, the place must **not have any** tokens
- If it has a lector arc, the place must have **at least one token**.
- If it has a guard, the guard value must be **true**.
- If it has events, the event queue must have **at least one event.**

Some considerations

- If it has an inhibitor arc, the place must **not have any** tokens
- If it has a lector arc, the place must have **at least one token**.
- If it has a guard, the guard value must be **true**.
- If it has events, the event queue must have **at least one event.**
- If it has a tag with a time lapse, the counter's value must be contained within the specified time lapse.

Some considerations

From the previous considerations it can be seen that, in order to fire a transition, a logic conjunction among all of the enumerated conditions and, the E vector of enabled transitions, is required. So, we define some auxiliary vectors

- **Vector of transitions disabled by inhibitor arcs (B)** is a binary vector of dimensions $m \times 1$, which shows which a zero which transitions are **disabled** by inhibitor arcs and with a one those that are not. It is obtained as:

$$\mathbf{B} = \mathbb{H} \cdot \mathbf{Q}$$

Where $\mathbb{H}$ is a matrix of dimensions $m \times n$, the incidence matrix that relates the places that are connected to transitions through inhibitor arcs. The elements $h_i j$ from the matrix have a value of **one** if there's an arc that runs from place p_i to transition t_j, and a value of zero if there's not. And **Q** a binary vector of dimensions $n \times 1$ where the element q_i have a value of **zero** if the marking on place p_i is different than zero, and one otherwise.

Some considerations

- **Vector of transitions disabled by lector arcs (L)** is a binary vector of dimensions $m \times 1$, that indicates with a zero which transitions are disabled by a lector arc and with a one the transitions that are not. It's obtained as:

$$\mathbf{L} = \mathbb{R} \cdot \mathbf{W}$$

Where $\mathbb{R}$ is a matrix of dimensions $m \times n$, the incidence matrix that relates the places that are connected to transitions through inhibitor arcs. The elements r_ij from the matrix have a value of one if there's a lector arc that runs from place p_i to transition t_j and zero otherwise. And **W** a binary vector of dimensions $n \times 1$ where the element w_i have a value of **one** if the marking on place p_i is different than zero, and zero otherwise.

Some considerations

- **Vector of transitions disabled by guards (G)** is a binary vector of dimensions $m \times 1$, that indicates with a zero which transitions are disabled by a guard and with a one the transitions that are not. It's obtained directly from the guards.

Some considerations

- **Vector of transitions disabled by guards (G)** is a binary vector of dimensions $m \times 1$, that indicates with a zero which transitions are disabled by a guard and with a one the transitions that are not. It's obtained directly from the guards.
- **Vector of transitions disabled by events (E)** is a binary vector of dimensions $m \times 1$, that indicates with a zero which transitions are disabled because there are not any events requesting to fire the associated transition. The v_i components of the V vector one if there is at least one event in the event buffer associated with transition t_i , and zero otherwise.

Some considerations

- **Vector of transitions disabled by time (Z)** is a binary vector of dimensions $m \times 1$, that indicates with a zero which transitions are disabled because the starting time has not been reached or because the time lapse has passed since the transition was first enabled. Otherwise its value is one, and is obtained as:

$$\mathbf{Z} = Tim(q(E, B, L, G, clk), intervals)$$

Some considerations

- **E** is the *enabled vector*, **B** is the *disabled by inhibitor* arcs vector, **L** is the *disabled by lector* arcs vector and **G** is the *disabled by guards* vector.

Some considerations

- **E** is the *enabled vector*, **B** is the *disabled by inhibitor* arcs vector, **L** is the *disabled by lector* arcs vector and **G** is the *disabled by guards* vector.
- The relation $Tim(q, intervals)$ is a binary vector of dimensions $m \times 1$. The value of the i component is one if the value of q_i , which is a counter, is within the time lapse indicated by the component $Inteval_i = [\alpha_i, \beta_i]$. Otherwise it's zero.

Some considerations

- **E** is the *enabled vector*, **B** is the *disabled by inhibitor* arcs vector, **L** is the *disabled by lector* arcs vector and **G** is the *disabled by guards* vector.
- The relation $Tim(q, intervals)$ is a binary vector of dimensions $m \times 1$. The value of the i component is one if the value of q_i , which is a counter, is within the time lapse indicated by the component $Inteval_i = [\alpha_i, \beta_i]$. Otherwise it's zero.
- For the q_i counter to start, the equation $e_i \wedge b_i \wedge l_i \wedge g_i$ must return a value of one. Otherwise the counter is set to zero. This counter is incremented one time-unit per clock cycle of the base time unit (clk).

Some considerations

- **E** is the *enabled vector*, **B** is the *disabled by inhibitor* arcs vector, **L** is the *disabled by lector* arcs vector and **G** is the *disabled by guards* vector.
- The relation $Tim(q, intervals)$ is a binary vector of dimensions $m \times 1$. The value of the i component is one if the value of q_i , which is a counter, is within the time lapse indicated by the component $Inteval_i = [\alpha_i, \beta_i]$. Otherwise it's zero.
- For the q_i counter to start, the equation $e_i \wedge b_i \wedge l_i \wedge g_i$ must return a value of one. Otherwise the counter is set to zero. This counter is incremented one time-unit per clock cycle of the base time unit (clk).
- The $intervals$ matrix of dimensions $m \times 2$, contains the lower and upper limit of the time frame in which the transition's firing is enabled.

Some considerations

- **Vector of transitions reset (A)** an integer vector of dimensions $m \times 1$, which contains the current marking value of the place that will be set to zero, while other components have a value of one. The $\mathbb{R}_{\mathbb{E}}$ matrix mathematically expresses reset arcs. The arc that runs from p_i to transition t_j is indicated as a value of one in the re_{ij} component of the matrix, otherwise is zero.

$$\mathbf{A} = Marking(\mathbb{R}_{\mathbb{E}} \cdot \mathbf{M}_j)$$

Some considerations

- The product results in a vector which components are zero if there's no reset arc, or the proper marking if there actually is a reset arc. The relation $Marking()$ results in a value of one if the component is zero, otherwise results in the value of the component.
- The **A** vector is multiplied, element by element, with σ the firing vector. This operation is indicated with $\#$. The σ vector has a one in the transition that we want to fire, therefore we obtain the quantity of shots necessary to take all of the tokens from the place to be reset, otherwise we obtain just one shot.

The final equation

- $\mathbf{E}_x$, the extended enabled vector, is obtained through the logic conjunction of all of the previous vectors. That it's:

$$\mathbf{E}_x = E \wedge B \wedge L \wedge V \wedge G \wedge Z$$

- To introduce the reset arc, it is necessary to multiply element by element $(\#)$ the vector that results in this conjunction with the vector **A**. This way, the extended state equation results in:

$$M_{j+1} = M_j + I \cdot (\sigma \wedge E_x) \# A)$$

Notes about timed and colored petrinets

- Time Petri nets (TPN) are classic Petri nets where each transition t is associated with a time interval $[\alpha_t, \beta_t]$. When t becomes enabled, it cannot fire before at time units have elapsed, and it has to fire no later than bt time units after being enabled, unless it has meanwhile become disabled by the firing of another transition.
- Here α_t and β_t are relative to the point in time when t last became enabled. The time α_t is the earliest possible firing time for t and is called earliest firing time of t, and β_t is the latest possible firing time for t and is called latest firing time of t.
- With the concept of time, we can also introduce the concept of probabilistic defining the **Stochastic Petri Nets** (SPNs) which were introduced in 1980 as a formalism for the description of Discrete Event Dynamic Systems (DEDS) whose dynamic behaviour could be represented by means of continuous-time homogeneous Markov chains.

Notes about timed and colored petrinets

- Another important type of PNs are the **Coloured PNs (CPNs)** is a graphical language for constructing models of concurrent systems and analysing their properties. CPNs is a discrete-event modelling language combining the capabilities of PNs with the capabilities of a high-level programming language. PNs provide the foundation of the graphical notation and the basic primitives for modelling concurrency, communication, and synchronisation.

The monitor and the relationship with PNs

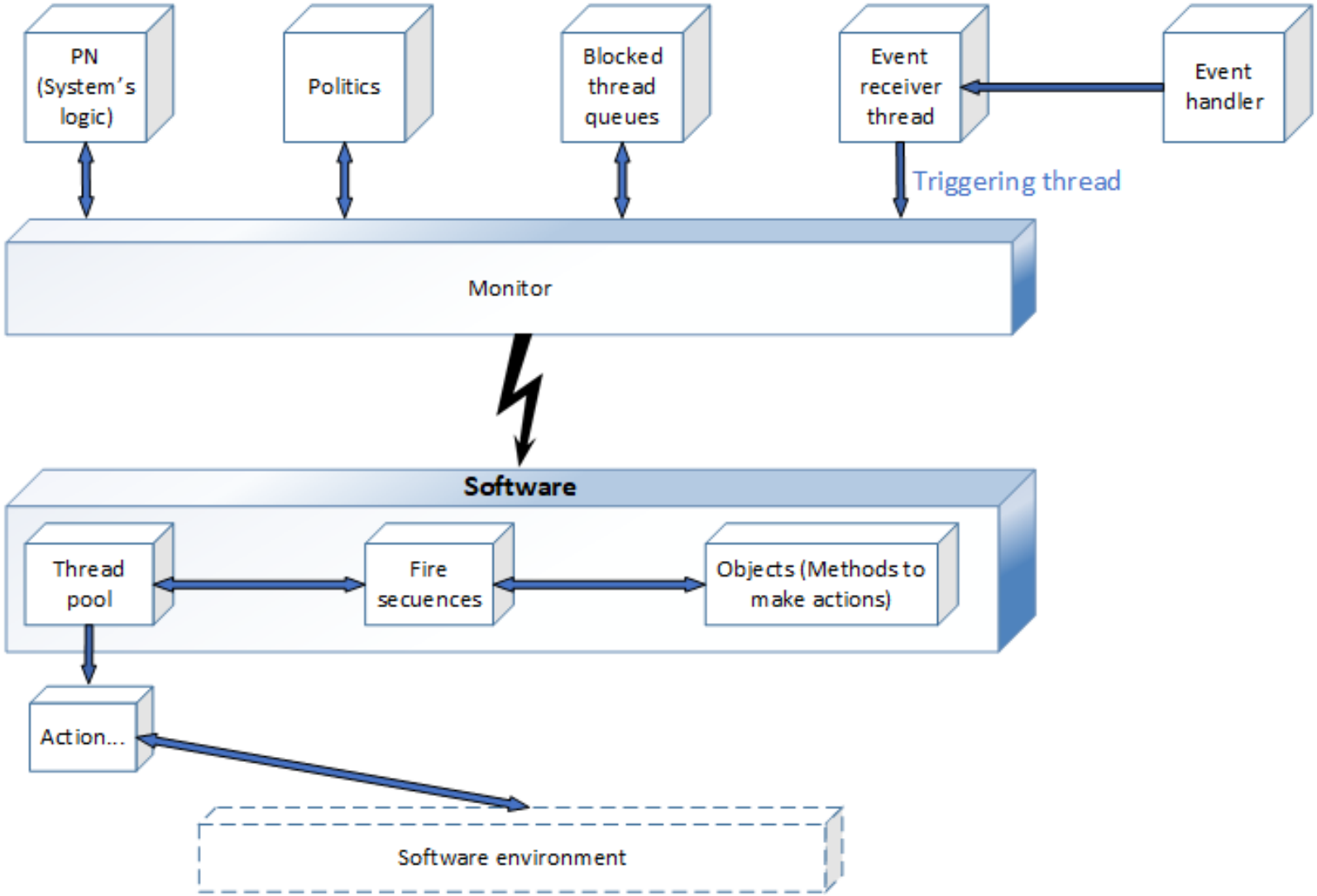


Figure: Block diagram Monitor

Algorithms made on C++

- One of the main advantages of formal models is that they enable us to define the behavior of a system unambiguously, develop algorithms for verifying properties and integrate them in a dedicated software tool.
- The firing rule of Petri nets associates a (finite or infinite) reachability graph with a net. This graph constitutes a formal representation of the net's behavior.

Algorithms made on C++

- PNs is a powerful tool, so we need powerful simulator to study our models. There is many simulators of PN, some writing in Java, others in Python, and possible some in C++. This is my simulator writed in C++.
- This program, written using the std 20 C++ also allow us to produce the the minimal coverability graph for a PN.

Q&A

Thanks for yuor time :)

Q&A

Thanks for yuor time :) Any questions ?

Twitter: @gabriel___val
Github: GabrielEValenzuela