SIMULATION - PRODUCTION SYSTEMS PETRI NETS - ARENA

All models are wrong but some are useful
Georges E.P. Box, statistician 1919-2013

If a problem has a solution, there is no need to worry.
But if it doesn't, then worrying doesn't change anything.
Tibetan proverb

Jean-Louis Boimond
University of Angers

Table of contents

I INTRODUCTION TO SIMULATION 4

I.1 MODEL BUILDING 5
I.2 LIMITS OF SIMULATION 6
I.3 DISCRETE EVENT SYSTEMS 6
I.4 SIMULATION OF PRODUCTION SYSTEMS 7
I.5 USE OF COMPUTER TECHNOLOGY 8

II ELEMENTS OF PROBABILITY AND STATISTICS 8

II.1 CONTINUOUS RANDOM VARIABLES 10
II.2 STANDARD PROBABILITY DISTRIBUTIONS 11
II.3 DISCRETE RANDOM VARIABLES 13

III SOURCES OF INPUT DATA 15

III.1 EXISTING DATA 15
III.2 PARTIALLY KNOWN DATA 17

IV MODEL VERIFICATION AND VALIDATION 17

IV.1 VERIFICATION 17
IV.2 VALIDATION 18

V INTERPRETING SIMULATION OUTPUT 19

V.1 ANALYSIS OF TERMINATING SYSTEMS 20
V.2 ANALYSIS OF NON-TERMINATING SYSTEMS 21

VI BASIC NOTIONS ON PETRI NETS 23

VI.1 GENERALITIES 23
VI.2 EVENT GRAPHS 25
VI.3 EXAMPLES 25
VI.4 OTHER CLASSES OF PETRI NETS 26

VII SIMULATION USING ARENA 28

VII.1 BASIC NOTIONS 28
VII.2 BLOCKS TO BUILT MODEL 32
VII.3 BLOCKS TO ANALYZE MODEL 45
VII.4 GRAPHIC ANIMATION 49
VII.5 INPUT ANALYZER 51
VII.6 OUTPUT ANALYZER 53

Bibliography

- *Discrete Event Systems - Modeling and Performance Analysis*, Christos G. Cassandras, Aksen Associates Incorporated Publishers, ISBN 0-256-11212-6.
- *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, J. Bank, Wiley Interscience, 1998.
- *Introduction to Simulation Using SIMAN*. Second Edition, C. Dennis Pegden, R.E. Shannon, R.P. Sadowski, Ed. Mc Graw-Hill.
- *Simulation Modeling and Analysis with ARENA.* T. Altiok, B. Melamed, Elsevier, 2007.
- *Optimisation des flux de production : Méthodes et simulation*, A. Ait Hssain, Ed. Dunod, 2000.
- *Du Grafcet aux réseaux de Petri*. R. David, H. Alla, Hermès, 1989.
- Cours de « *Simulation informatique des systèmes de production* », P. Castagna, A. L'Anton, N. Mebarki, 97/98 - IUT OGP Nantes.
- Cours de « *Réseaux de files d'attente et simulation* », J. P. Chemla, 96/97 - Université de Tours.
- *Probabilités et statistiques*. 3ème édition, A. Ruegg, Presses Polytechniques Romandes.

ARENA: blocks, probability distributions, variables

ASSIGN block **37**
BATCH block **41**
COUNTER block **45**
CREATE block **32**
DECIDE block **39**
DELAY block **34**
DISCrete distribution **14**
DISPOSE block **33**
EXPOnential distribution **12**
MATCH block **40**
NORMal distribution **13**
NQ variable **48**
NR variable **48**
PROCESS block **43**
QUEUE block **35**, **41**
RECORD block **45**
RELEASE block **36**
RESOURCE block **36**
SEIZE block **34**
SEPARATE block **42**
STATION block 50
TALLY block **46**, **48**
TIME PERSISTENT block **48**
TRIAngular distribution **11**
UNIForm distribution **10**
VARIABLE block **38**

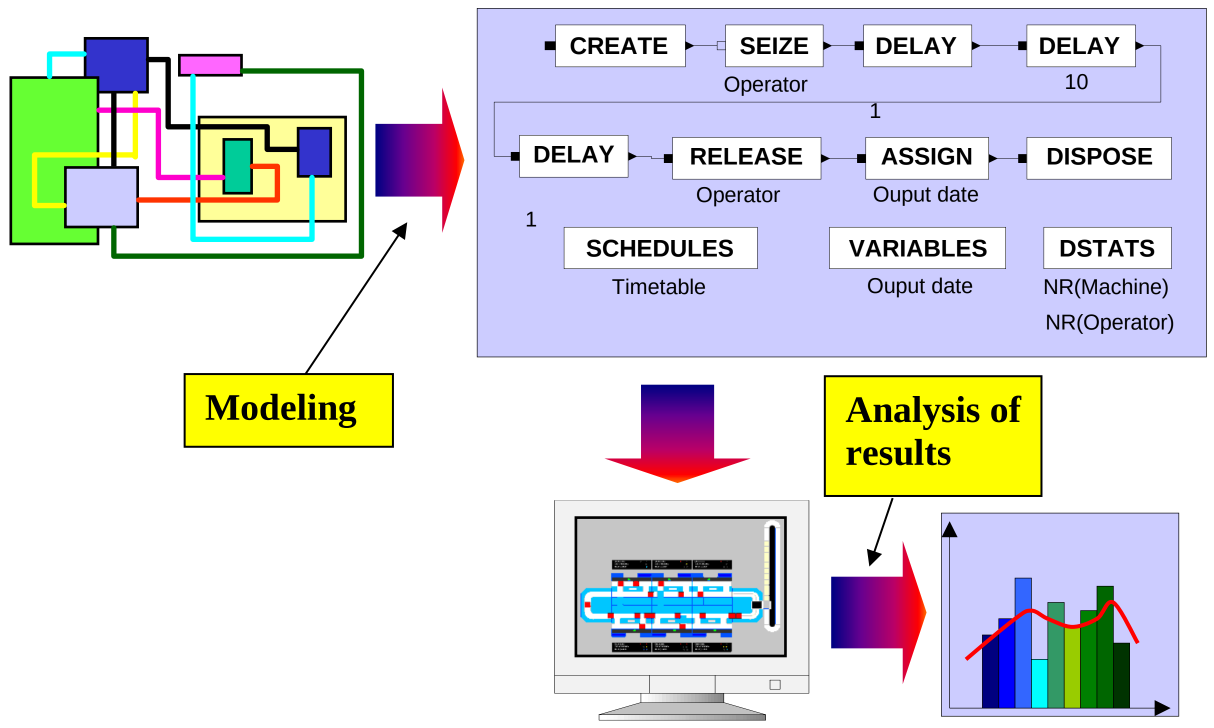
I INTRODUCTION TO SIMULATION

A large number of (real) systems are complex in the sense that they are very/too difficult/ to study *via* analytical models due to the lack of computable solution. **Simulation** is a technique to study the dynamic behavior of complex system based on the building and analyze of *software model* ('modèle logiciel') of the system.

Simulation is a process that consists of:

- *build a model* of the (real) system to study,
- *conduct experiments* on the model (no calculation),
- *interpret the observations* provided by the model execution and *make decisions* about the system.

The goal can be to understand the dynamic behavior of a system, to compare configurations, to evaluate different control strategies, to analyze and optimize performances.



Fields of application are various. some examples of classic problems related to these fields are listed below.

- *Workflow systems*
  - balancing of assembly lines,
  - design of transfer systems between stations,
  - sizing ('dimensionnement') of workshop stock,
  - comparison between production line managements.
- *Logistics flows and transport systems*
  - design and sizing of warehouse,
  - sizing of trucks fleet,
  - study of controlling procedures for the flow of vehicles in circulation.
- *Production of services*
  - study of banking transactions,

- management of fast-food restaurant,
- comparison between aircrafts maintenance policies.

▪ *Computer systems and telecommunications*
  - study of server memory queue,
  - study of behavior of users,
  - design and sizing of *hub*.

▪ *Other classes of applications*
  - military field (logistical support, coordination of operations, etc.),
  - hospital management (staff, beds, emergency department, etc.),
  - nuclear, weather forecasting, games, etc.

General methodology

Four phases are classically considered during the simulation process: Analysis and modeling of the system (leading to a *conceptual model*, this model being also conditioned by the given objective), implementing a computer program equivalent to the conceptual model (leading to a *software model*, also called *simulation model*), experimentation and interpretation of the results leading to decisions or actions on the system.

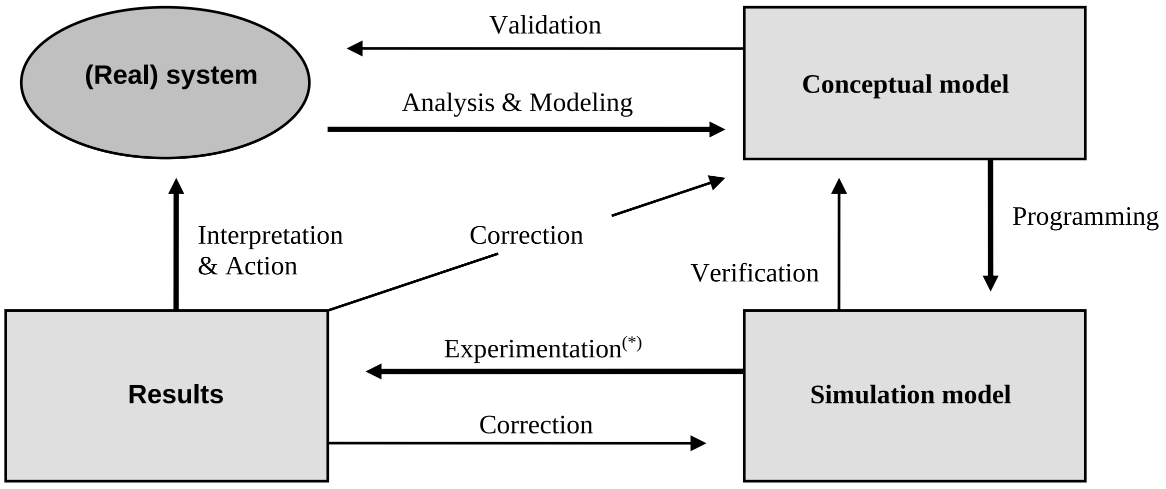


Figure 1: Simulation methodology.

(*) *Experimentation*: It is about building theories, putting forward hypotheses (‘d’avancer des hypothèses’), based on observed behavior of simulation model.

Here we will represent conceptual models by using *Petri nets*, *cf.* Chp. VI; simulation model, as well as experiments, will be realized by using ARENA software, *cf.* Chp. VII.

I.1 MODEL BUILDING

Model building is an essential step during simulation process in the sense that the quality of the results obtained at the end of experiments is mainly dependent on the quality of the modeling. Different points need to be considered:

- Define the objective of the modeling (related to the specifications): Why do we model? What are we studying? What do we want to do or improve?

- Define the limits of the system (identify inputs and outputs) and the elements (*via* the realization of function or process) composing the system.
- Define the interactions between these elements (hierarchy).

I.2 LIMITS OF SIMULATION

Simulation is not an *optimization technique* in the literal sense, it can only establish the performance of a solution designed and imagined by the simulation designer. Simulation is an *iterative technique* that does not propose final solution but only allows the designer to consider possible choices. In any case, it is the designer who will have to decide what is the best answer to the given problems.

Simulation results are often complex to interpret. Random phenomena are studied, and analysis techniques require rigor; it is often difficult to distinguish between the crucial and the anecdotal (model must be neither too coarse nor too precise), all this to be done in an often constraining deadline.

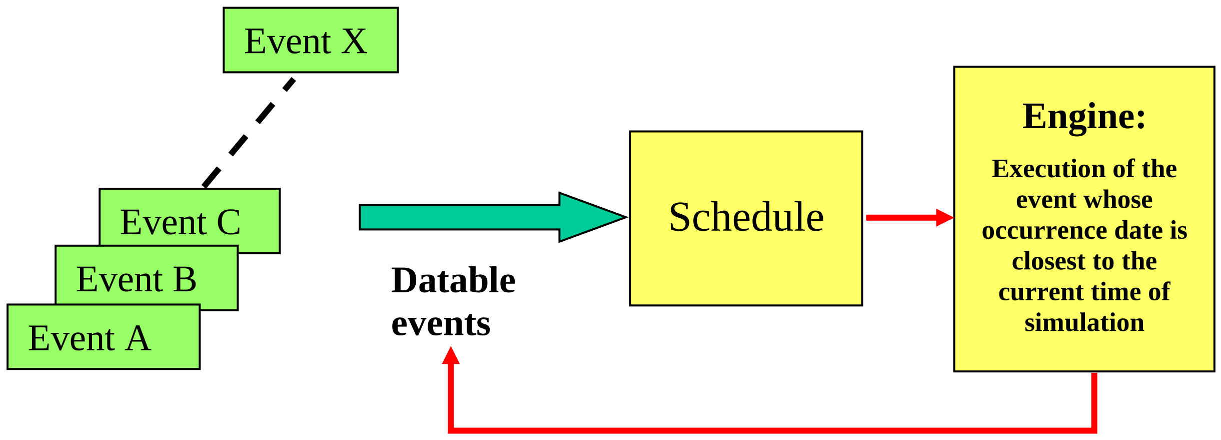
I.3 DISCRETE EVENT SYSTEMS

The systems we will consider are *discrete event systems* which means that they are represented by discrete event models. The *state* space is governed by *discrete events* in the sense that transitions between states are associated with the occurrence of asynchronous discrete events. The changes of state of these systems (occurring at discrete moments over time) are instantaneous. For example, if a variable represents the number of workpieces in a stock, its value only changes when workpieces enter or leave the stock.

Production systems, traffic systems (air, rail, naval, etc.), communication systems, computer systems are examples of discrete event systems, that is, are governed by discrete events, some of one being caused voluntarily (departure of a train, pressing a key on a keyboard) and others not (breakdown of equipment).

Model reproduces the evolution over time of the state[1] of system under the effect of activities that are realized. The evolution of event-driven simulation is done through the management of a schedule: the model of system goes from one state to another state following the triggering (‘déclenchement’) of an ***event***. Each event is associated with a ***function to be executed*** which can modify the state of the system through the triggering of one or more events.

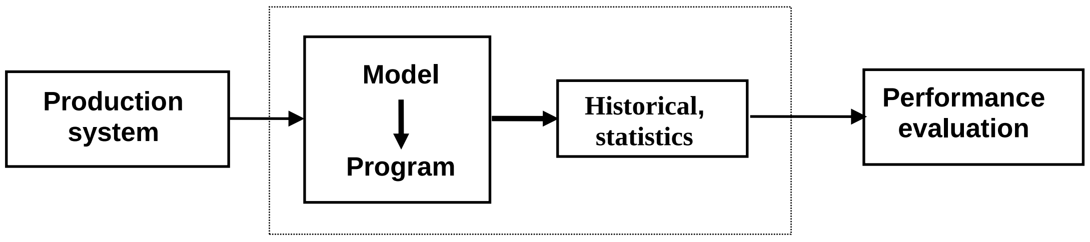
[1] The evolution of state is governed by discrete events unlike continuous systems (governed by differential equations) where state evolves *continuously* over time.


Event X
Event C
Event B
Event A
Datable events
Schedule
Engine:
Execution of the event whose occurrence date is closest to the current time of simulation

I.4 SIMULATION OF PRODUCTION SYSTEMS

Production system consists of an *operating* system (physical), a *control* system (control part) and an *information* system linking the latter two. It is crossed by a *flow of information* (presence of workpiece, state of machine) and a *physical flow* (raw material, workpiece). The system to simulate can be existing, to be modified or not yet built.

Automated production systems are characterized by high complexity and flexibility. The simulation of these systems often requires global approach considering at the same time technical aspects (characteristics of production resources, storage capacities, geometry of transport network, etc.) and human aspects (social constraints, teamwork, overtime, etc.).



Model describes the functioning of system (its structure, its dynamic behavior) with the degree of detail necessary to solve the given problem. It allows a representation of product flows knowing that flow is:

- slowed down by activities mobilizing resources (after waiting for their availability) during a certain time (operating times, transfer time, etc.),
- constrained by operating rules (ranges, technological constraints),
- directed by the rules of conduct ('règles de conduite') given by the control system.

Historical and *statistics* concern information about transportation (transport time of workpieces from one place to another, etc.), resource commitment rates ('taux d'engagement des ressources'), queue lengths, etc.

Performance evaluation, in terms of product flows, exploits this data for:

- determine absolute performance (production volume, maximum cycle time),
- predict performance under certain conditions (presence of failures),
- do a sensitivity analysis (among similar choices),
- compare alternatives (among possible choices).

Performance evaluation is often based on production rate (average number of workpieces per unit of time), WIP (Work In Progress/Process, total number of workpieces in system at any given time), makespan (time interval between the start and end of workpieces production).

These performance indicators are then aggregated for decision-making related to design assistance (‘aide à la conception’), system control, etc.

I.5 USE OF COMPUTER TECHNOLOGY

Three approaches are usually used to perform a simulation:

1. Write from scratch the program corresponding to the given problem and system by using standard computer languages such as Python, Java, C#. Implementation is often time consuming, but there is a lot of flexibility.

2. Development of simulation model is realized through a program written by the simulation designer from modeling primitives provided by the *simulation languages*. Such languages offer great flexibility during model development phase. For example, ARENA (one of the main standard simulation software in France) offers modeling primitives particularly adapted to production systems (resource modeling and transport primitives).

3. Use a *simulator* dedicated to a particular type of system and problem. Model is given, and it is enough to parameterize it to adapt it to the studied case. This alternative has the advantage that no programming is required (only parameters must be entered), but it is not always easy to find the right software for the system and problem.

II ELEMENTS OF PROBABILITY AND STATISTICS

Many real-life systems exhibit behavior with random phenomena. Whatever the power of computers, all possible deviations of such systems cannot be simulated, the statistical tool is an alternative to study and control the consequences of these random variations on system behavior.

Modeling of random discrete event systems simply means that randomness (‘l’imprévu’) is introduced into events through two basic ways: event occurrence times or event state transitions may be random.

Probability theory makes it possible to model and study random phenomena, we speak about random events, random variables, probability distributions, etc.

For example, in a production system, many phenomena are random such as:
- operating time of manual operation,
- life time of tool,

- absenteeism of operators,
- period of arrival of production orders starting the production.

Statistics are based on the observation of concrete phenomena. Purpose is to collect, process and interpret observational data, we speak about population of individuals, characteristic variables, samples ('échantillons'), averages, etc.
Probabilistic models make it possible to approximate the observed data (imprecision, errors, distribution in the population) as random variables according to a certain probability distribution → *simplifying models*.

As the sample is drawn at random ('tiré au hasard'), the characteristics of the data to be processed are *random variables* → application of probability theorems such as the central limit theorem[2].

Simulation uses the results of probability-statistics essentially to:
- approach empirical data through probability distributions
  → functions integrated into simulation model (probability distributions),
- statistically interpret the data generated by model
  → mean, standard deviation, confidence interval, etc.

Definition of probability

We consider the set Ω of all possible eventualities of the considered random phenomena resulting from a test (experiment, observation, or simulation), each of these eventualities is called an *elementary* event. Any event is defined as a subset A of Ω containing all the elementary events of Ω composing the event A. The *probability* attached to an event A is a number $P(A)$, *probability* satisfies the following postulates:
- $\forall\ A \subseteq\ \Omega$, we have $0 \leq P(A) \leq 1$, in particular $P(\emptyset) = 0$ (probability of empty set) and $P(\Omega) = 1$ (probability of event Ω),
- $\forall\ A, B \subseteq\ \Omega$ satisfying $A \cap B = \emptyset$ (disjoint events), we have $P(A \cup B) = P(A) + P(B)$ (in other words, the probability of a set is equal to the sum of the probability of its disjoint subsets) which is a particular case of the equality $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

The problem of assigning probabilities to a set of events can be solved in several cases as follows:
- if elementary events are *finite*, a series of repetitions of test can be realized: The frequency of occurrence of each event provides an estimate of its probability,
- if events are in infinite number, we can define on this set a probability distribution.

[2] The mean of a sample size n extracted from any population of mean μ and standard deviation σ is distributed according to a practically normal distribution of mean μ and standard deviation $\frac{\sigma}{\sqrt{n}}$ when the sample size is large enough. For a population coming from a normal distribution, the central limit theorem is valid for all n. For other distributions, the larger the sample size, the closer the distribution is to the normal distribution. It can be considered that from n equal to 30, the mean of a sample is distributed in a substantially normal way.

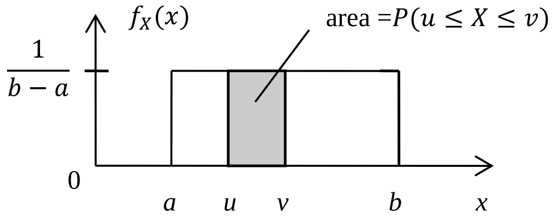
II.1 CONTINUOUS RANDOM VARIABLES

A continuous random variable X is a real-valued function defined on a set Ω (set of possible events) such that the set of values taken by X, denoted $X(\Omega)$, is a finite or infinite interval. For example, let be Ω equal to an interval $[a,b]$ representing all the possible values of the diameter of the workpieces manufactured by a turning machine ('tour d'usinage').

Example of the Uniform distribution (UNIF) continuous: Let be X a random variable that can take all values of a finite interval $[a,b]$, without privileging any region of $[a,b]$ (we talk about equiprobable events). Also, probability that X takes a value belonging to the interval $[u,v]$ ($\subset [a,b]$) is proportional to its length, that is,

$P(u \le X \le v) = (v-u)/(b-a)$,

that is $P(u \le X \le v) = \int_u^v f_X(x)dx$ where $f_X(x) = \begin{cases} 1/(b-a) & \text{if } a \le x \le b \\ 0 & \text{elseif} \end{cases}$.



$f_X(x)$ is a *probability density function*, it defines the random (stochastic) behavior of the random variable X and thus characterizes its probability distribution (often abbreviated to distribution).

The *Uniform* distribution (*distribution of maximum ignorance*) is used when one has no information except domain knowledge $[a,b]$.

$f_X(x)$ is a *probability density function* of the random variable X if, and only if,

$$\forall\, a,b \in R^2, P(a \le X \le b) = \int_a^b f_X(x)dx.$$

Remark: For continuous random variable, considering an event of the type '$X = x$' does not make sense (indeed, we have: $P(x \le X \le x) = 0$).

Probability density function $f_X(x)$ is such that:

$$\begin{cases} f_X(x) \ge 0,\ \forall\, x \in R, \\ \int_{-\infty}^{+\infty} f_X(x)\, dx = 1 \text{ (corresponding to the probability of the sure event} = 1). \end{cases}$$

Remark: $f_X(x)$ is continuous on $\mathbb{R}$ except (possibly) in a finite number of points (for example, the density of the uniform distribution is continuous, except in 2 points).

We define the *mean* M, also called *mathematical expectation* ('*espérance mathématique*') $E(X)$, by:

$M = \int_{-\infty}^{+\infty} x\ f_X(x)\ dx.$

We define the *variance* σ^2 ($\sigma^2 \ge 0$), also denoted $Var(X)$, by:

$\sigma^2 = \left(\int_{-\infty}^{+\infty} x^2 f_X(x)\ dx\right) - M^2$, also equal to $\int_{-\infty}^{+\infty} (x-M)^2 f_X(x)\ dx$.

N.B.: The ***mean*** is a *position parameter* that provides information on the order of magnitude of the values taken by the random variable X. ***Variance*** is a *measure of the dispersion* of these values around their mean. The smaller the variance (≥ 0), the more the values taken by X are concentrated around the mean.

Example: In the case of the previous uniform distribution, we have:

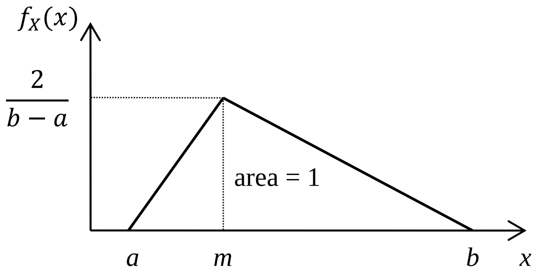
$$M = \int_a^b \frac{x}{b-a}\, dx = \frac{a+b}{2} \text{ and } \sigma^2 = \int_a^b \frac{x^2}{b-a}\, dx - \left(\frac{a+b}{2}\right)^2 = \frac{(b-a)^2}{12}.$$

The *standard deviation* (‘*écart type*’) is defined by $\sigma = \sqrt{\sigma^2}$.

Most of the random phenomena encountered in practice can be studied *via* a limited number of distributions. Let us briefly recall the most commonly used ones.

II.2 STANDARD PROBABILITY DISTRIBUTIONS

a) TRIANGULAR DISTRIBUTION (TRIA)



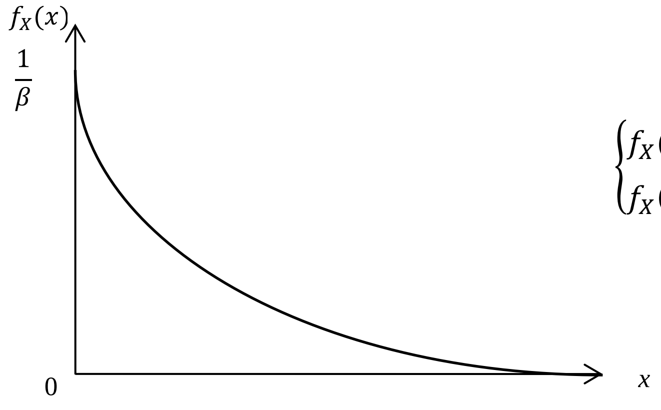
$$\begin{cases} f_X(x) = \frac{2(x-a)}{(m-a)(b-a)} & \text{if } a \leq x \leq m, \\ f_X(x) = \frac{2(b-x)}{(b-m)(b-a)} & \text{if } m \leq x \leq b, \\ f_X(x) = 0 & \text{elseif.} \end{cases}$$

$$D = [a, b] \;;\; M = \frac{a+m+b}{3} \;;\; \sigma^2 = \frac{a^2+m^2+b^2-am-ab-mb}{18}.$$

Application: This distribution is used when we have an estimate of minimum, maximum, and the most probable value.

Exercise: Let $a = 0, m = 2, b = 3$, calculate $P(1 \leq X \leq 2{,}5)$.

b) Exponential Distribution (EXPO)



$$\begin{cases} f_X(x) = \frac{1}{\beta} e^{-x/\beta} & \text{if } x > 0 \quad (\beta > 0), \\ f_X(x) = 0 & \text{elseif.} \end{cases}$$

$D = \ [0, +\infty\ [\ ;\ M = \beta\ ;\ \sigma^2 = \ \beta^2.$

Application: This distribution is often used in practice. For example, in the case of time separating the arrivals of 2 successive ‘customers’ in the study of a waiting phenomenon, or in the case of the operating time of a technical equipment.

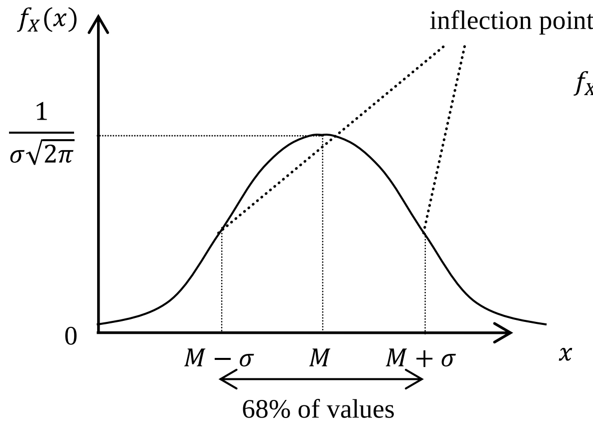
The *exponential* distribution is the *only continuous* distribution that allows the consideration of phenomena *without memory*. Indeed, the probability that X is greater than, or equal, to $x + x_0$, knowing that X is greater than, or equal, to x_0, depends on the value of x, and is independent of the value of x_0, that is: $P(X \geq x + x_0 \mid X \geq x_0) = P(X \geq x)$.

For example, it is often accepted that the lifetime T of an electronic device is specified as an exponential distribution. Also the probability of correct functioning of the device in a time interval $[\Delta_0, \Delta_0 + \Delta]$, that is, $P(T \geq \Delta + \Delta_0 | \ T \geq \Delta_0)$, depends only on the length of this interval, and not on its position relative to the time axis, that is: $P(T \geq \Delta + \Delta_0 \mid T \geq \Delta_0) = P(T \geq \Delta)$.

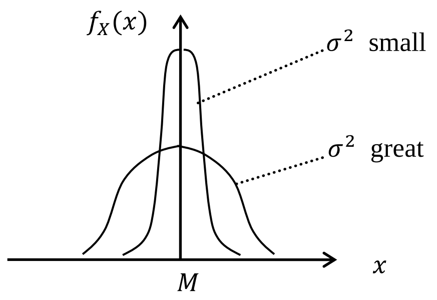
Proof: Let be the event A corresponding to the fact that $X \geq x_0$ and the event B corresponding to the fact that $X \geq x_0 + x$. We have $P(X \geq x_0) = \int_{x_0}^{+\infty} f_X(x)\ dx$ and $P(X \geq x_0 + x) = \int_{x_0+x}^{+\infty} f_X(x)\ dx$. Hence $P(B \mid A)$ is equivalent to $P(X \geq x_0 + x \mid X \geq x_0)$.
Knowing that $P(A \cap B) = P(A) \times P(B \mid A) = P(B) \times P(A \mid B)$ (conditional probability), we have $P(B \mid A) = \frac{P(B) \times P(A \mid B)}{P(A)}$.
Knowing that $P(A \mid B)$ is equivalent to $P(X \geq x_0 \mid X \geq x_0 + x) = 1$, we have $P(B \mid A) = \frac{P(B)}{P(A)}$.

Hence $P(B \mid A) = \frac{P(B)}{P(A)} = \frac{P(X \geq x_0 + x)}{P(X \geq x_0)} = \frac{\int_{x_0+x}^{+\infty} \frac{1}{\beta} e^{-\frac{u}{\beta}}\ du}{\int_{x_0}^{+\infty} \frac{1}{\beta} e^{-\frac{u}{\beta}}\ du} = \frac{-\left[e^{-\frac{u}{\beta}}\right]_{x_0+x}^{+\infty}}{-\left[e^{-\frac{u}{\beta}}\right]_{x_0}^{+\infty}} = \frac{e^{-\frac{(x_0+x)}{\beta}}}{e^{-\frac{x_0}{\beta}}} = e^{-\frac{x}{\beta}}$ which is a function of x only (independent of x_0).

c) Normal Distribution (NORM)



$f_X(x)=\frac{1}{\sigma\sqrt{2\pi}}\ e^{-\ (x-M)^2/2\sigma^2}$



$D\ =\ ]-\infty,+\infty[$; mean $=M$; variance $=\sigma^2$.

Application: This distribution applies in the case of systems whose distribution is symmetrical and for which the mean and standard deviation are estimated (for example, the variations in the length of workpieces).
This distribution also allows to model a data that is the sum of a large number of random data (central limit theorem).

N.B.: Instead of probability density $f_X(x)$, one can use the distribution function $F_X(x)$ to characterize the distribution of a random variable X.

We have $F_X(x)=P(X\leq x)=\int_{-\infty}^{x} f_X(u)\ du$ for $-\infty<x<+\infty$.

$F_X(x)$ is a continuous, monotonous increasing function, such that $F_X(-\infty)=0$ et $F_X(+\infty)=1$, $F_X^{'}(x)=f_X(x)$. It allows to calculate probabilities of the form $P(a<X\leq b)$ without performing an integration (which is the case by using $f_X(x)$); indeed $P(a<X\leq b)=F_X(b)-F_X(a)$.

II.3 Discrete Random Variables

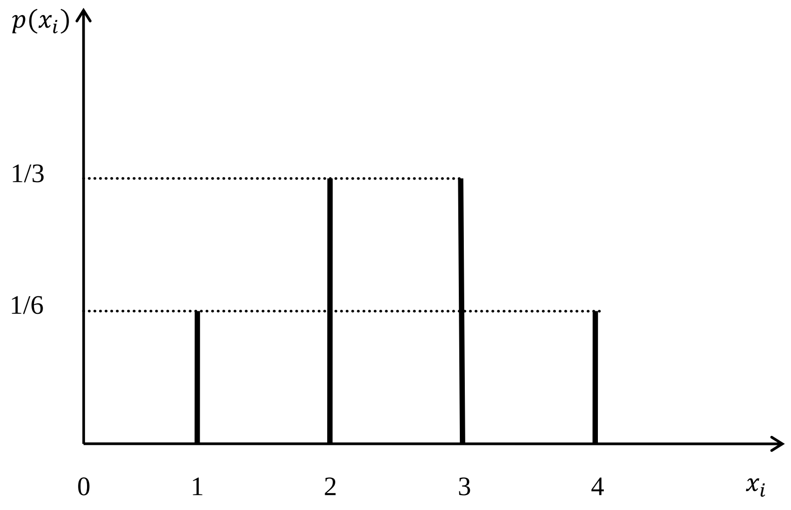
A random variable is *discrete* if it can only take a *finite* number of values (for example: $\Omega=\{stack, face\}$ in the case of throwing a coin). For each value x_i, we associate the probability $p(x_i)$ of occurrence of this value.

For N values, the set of associated probabilities is such that:

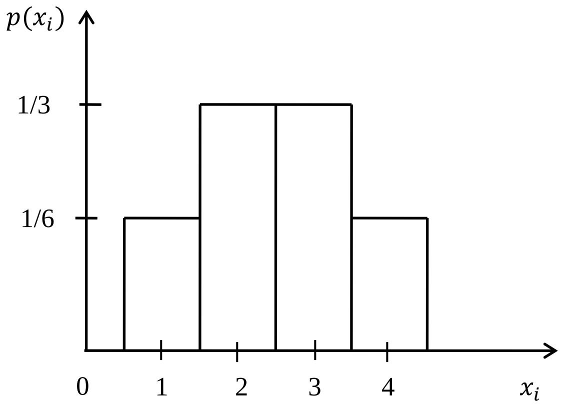
$$\sum_{i=1}^{N}\ p(x_i)\ =\ 1 \quad \text{if } N \text{ represents the set of values.}$$

Example: We define a system capable of producing four types of products denoted 1, 2, 3, 4. When the production orders arrive, we know that the probability of having a product 1 is equal to $1/6$, that of having a product 2 is equal to $1/3$, that of having a product 3 is equal to $1/3$ and that of having a product 4 is equal to $1/6$.

The distribution is represented either by the following *bar graph* (‘*diagramme en bâtons*’) indicating $p(x_i)$ as a function of x_i:



or by a *histogram*[3]:



Définitions

The *mean* (arithmetic) M is equal to $\sum_{i=1}^{N} x_i p(x_i)$.

Exercise: Calculate the average considered in the previous example.

The variance σ^2 is equal to $(\sum_{i=1}^{N} x_i^2 p(x_i)) - M^2$, also equal to $\sum_{i=1}^{N}(x_i - M)^2 p(x_i)$.

The *cumulative probability* (a concept used in ARENA) is defined by

$$p_c(x_i) = \sum_{l=1}^{i} p(x_l).$$

In the previous example, we have $p_c(x_1) = \frac{1}{6},\ p_c(x_2) = \frac{1}{2},\ p_c(x_3) = \frac{5}{6},\ p_c(x_4) = 1$.

Let be $x_1, \cdots, x_n$ a set of n possible discrete values, the **discrete empirical distribution** $\mathbf{DISC}(p_c(x_1), x_1, \cdots, p_c(x_i), x_i, \cdots, p_c(x_n), x_n)$ is such that it returns the value x_i with a *cumulative* probability equal to $p_c(x_i)$. By construction, we have: $p_c(x_1) = p(x_1)$ and $p_c(x_n) = 1$. For example, the distribution $\mathbf{DISC}(0.3,1,0.4,2,1,4)$ returns: the value 1 with a

[3] A set of rectangles of the same width whose surfaces are proportional to probabilities $p(x_i)$.

probability equal to 0.3; the value 2 with a probability equal to $0.1(= 0.4 - 0.3)$; the value 4 with a probability equal to $0.6(= 1 - 0.4)$.

Application: Discrete random variables apply in the case of direct injection of empirical data in model (for examples: type of workpieces, batch size).

III SOURCES OF INPUT DATA

Data quality is as important as **model quality** (*garbage in - garbage out*); for example, in the case of production system, this can concern interarrival times of workpieces, processing times, travel times, scrap rates ('taux de rebut'), demand rates.

There are two main problems:

P1) Data collection
→ which? available? relevant? how to collect them?

P2) Stochastic systems
→ direct reading of empirical data or selection from an associated theoretical distribution?

The possible origins of data are of different nature:

- Historical records → databases to be request (update problems ('pb de mise à jour')),
- Observational data → human resources (mistakes, neglect of extremes and forgetting the past),
- Similar systems → pay attention to inferences,
- Information given by the vendor/designer (often optimistic).

Two cases are to be considered: the system data (average, minimum, maximum, etc.) are available/existing or partially known.

III.1 EXISTING DATA

Since **the P2** problem does not have a clear answer, simulation software often offers both possibilities.

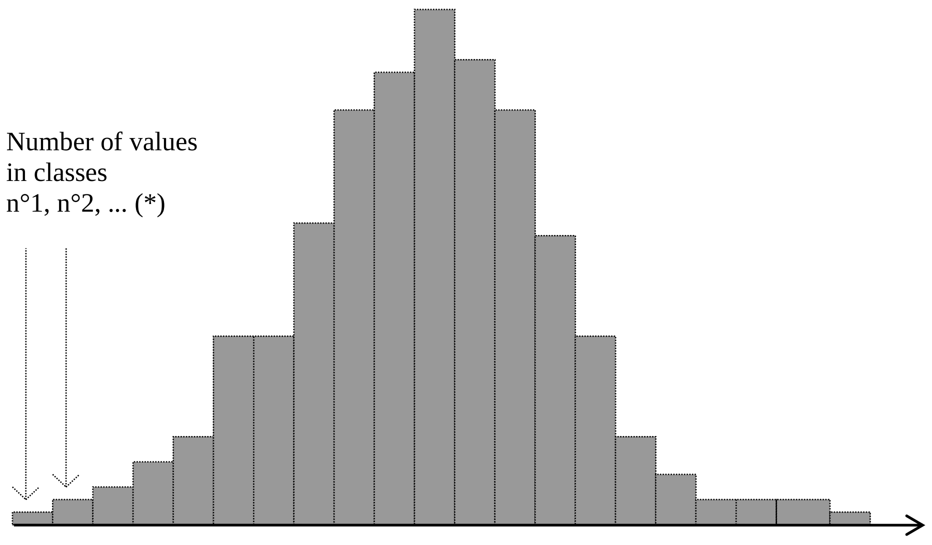
It is often interesting for theoretical and practical reasons to be able to describe a probability distribution by a theoretical distribution, which amounts to *analytically* expressing the probabilities $p(x_k)$ as a function of the index k. Well-known methods of mathematical analysis can then be applied to the calculation of probabilities, thus avoiding tedious numerical calculations.

- If empirical data are used directly, they are entered as cumulative empirical distributions (*frequency histogram*: grouping observations into classes, number of classes = $0\sqrt{n^{bre}\ d'observations}$),
- If we want to draw from the theoretical distributions ('faire des tirages à partir de distributions théoriques'), we need to:
  a) Choose distribution according to the shape of the data histogram,

b) Estimate its parameters,
c) Test the hypothesis (does the distribution match the data?).

Step *a*) is performed knowing the characteristics of standard distributions and visually comparing theoretical distribution and empirical distribution (frequency histogram).
Step *b*) involves the use of conventional estimators.
Step *c*) can be performed visually or using statistical tests of hypotheses (*Khi-deux*, *Kolmogorov-Smirnov*).

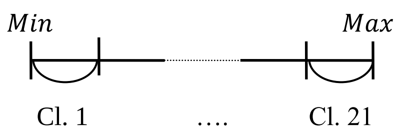
Example: We are interested in the processing time of a machine. We have a set of 500 values representing the time interval (obtained using a stopwatch) between each appearance of a workpiece at machine output. The machine input is assumed to be always supplied. We consider 21 classes (of identical width) to construct frequency histogram.



REAL data Data pts =500 Intervals = 21 Range: -1 to 12
Mean = 5,02 StdDev = 1,88 Min = -0,4531 Max = 11,3

NORMAL DISTRIBUTION: NORM(5,02; 1,88)
Sq Error = 0,0008231

(*) Hypotheses: Min et Max are finite values.
A value $x \in Classe\ n$ with $1 \leq n \leq 21$ if and only if $Min + (n-1)\frac{Max-Min}{21} \leq x < Min + n\frac{Max-Min}{21}$.



We consider a class $[-\infty, \text{real value}[$, resp. $[\text{real value}, +\infty[$, if the value $Min = -\infty$, resp. $Max = +\infty$.

III.2 PARTIALLY KNOWN DATA

When systems do not yet exist, or their desired data (time, resources) are not available, we must use the estimation of operators, designers, equipment suppliers, etc to characterize the system variables. Three cases often arise:

1. Only the mean M is available

We can use:
- Directly M as the constant value of the variable if the dispersion (standard deviation) is small,
- An exponential distribution (large dispersion: high variability) of M if the nature of the phenomenon justifies it.

2. Min et Max are available

We can use:
- A uniform distribution of parameters Min and Max, that is, the distribution of *ignorance* if there is no reason to think that probabilities are not equiprobable,
- If the data are centered around mean $M = (Min + Max)/2$, one can apply a normal distribution centered around M with a standard deviation $\sigma = (Max - Min\ )/6$ if the data are numerous, and $\sigma = (Max - Min\ )/4$ elseif.

3. Min, Max and the most probable value m are available

We can use a triangular distribution of parameters Min, m, Max.

IV MODEL VERIFICATION AND VALIDATION

Simulation models can evolve significantly during their development (*scenario* testing, what happens if? …). The major difficulties are to know:
- How to trust the model?
- How to transmit it to the user?

Before extracting inferences from statistical results of a simulation model, it is necessary to ensure that it represents the system, which usually involves two steps: verification and validation.

IV.1 VERIFICATION

Verification consists of ensuring that the model works as the designer expected (without logical errors), which requires being able to isolate errors (the most difficult step) in order to correct them. Verification is made easier if we start with a simple model that we improve gradually. The following techniques (or behavior to have) are used to isolate errors:

1. Always consider that the model contains errors and look for them (destructive approach, rather than constructive one).

2. Involve people not involved in design and implementation.

3. Review the model and data with the help of at least one client and one language expert (in addition to the developer).

4. Perform tests:
- Replace random times with constants,
- Test only a part of the model,
- Test the model under boundary conditions. To do this:
  - Increase the arrival rate and/or decrease the service rate to create congestion, or create phenomena of 'famines' of machines,
  - Reduce the size of stocks to create saturation phenomena,
  - Modify the distribution of workpiece types (*job mix*) to increase the arrival of workpieces of less frequent types,
  - Increase the occurrence rate of less frequent events (e.g., failure).

5. Generate and analyze the trace of the model to check the path of the workpieces, the changes of state at the end of a wait (at the level of a queue, due to an activity, etc.).

6. Use animation (powerful technique).

7. Correct mistakes by identifying the real causes and not just treat the symptoms (logical reasoning remains the best approach).

8. Avoid classic mistakes, especially about:
- Input data acquisition (units of measurement),
- Initialization of the simulation model,
- Arithmetic errors (parentheses, type conversion, etc.).

IV.2 VALIDATION

Three *questions* need to be asked:
- Does the model correctly represent the (real) system?
- Is the behavior data generated by the model characteristic of the (real) system?
- Does the user trust the model results?

Three *points of view* must be considered:
- That of the developer,
- That of a person evaluating the model (supervisor, client),
- That of the end user (decision-maker).

Three *types of tests* can be applied:

1. Is the behavior reasonable?

- Continuity: Small changes in input parameters
  $\rightarrow$ small changes in state and output variables.

- Consistency: Almost identical executions, for example, when random generator is changed
  $\rightarrow$ almost identical results.

- Degeneration: Removing a component (a 'mode') from the model, for example, a machine is deleted.
  $\rightarrow$ effects on results

- Absurd conditions: Absurd input parameters, for example, increasing the marketing budget to infinity should not lead to infinite sales ('ventes')

$\rightarrow$ absurd results.

2. Testing the data and the model structure

Theories and assumptions must be correct, and the model representation must be adequate for the desired use.

$\rightarrow$ Validity of 'façade': The behavior seems correct for users familiar with the (real) system (logic, input-output).

$\rightarrow$ Verification of structure and boundaries: Correspondence between the conceptual model and the system of reference.

3. Testing the model behavior

It consists in studying the behavior of the model in relation to the system of reference.

$\rightarrow$ Comparison of behaviors: Statistical tests to compare results (*Khi-deux*, *Kolmogorov-Smirnov*, etc.).

$\rightarrow$ Generate symptoms:
- Does the model generate difficulties already known in the system?
- Does the model produce known results for given inputs?

$\rightarrow$ Behavior anomaly: Anomaly in the model can lead to discovery of equivalent anomaly in the (real) system?

$\rightarrow$ Behavior prediction: Model prediction versus system tests.

V INTERPRETING SIMULATION OUTPUT

Running a simulation model can generate:
- A simulation report including the means, standard deviations, minimums, and maximums of the observed variables, etc.
- A history of the evolution of these variables during the simulation.

The quality of the mean (arithmetic) as an estimator of the true mean depends, among other things, on the number of observations. Similarly, the standard deviation is biased for a small number of observations.

Such a simulation report is not enough to make credible conclusions about the performance of the system (changing the random number generator, without changing the model, is enough to generate different results). Graphic animation is also not enough. In fact, we are often satisfied with the simulation report and/or the animation, especially when the project is late.

The results generated by a model act as measurements on a sample ('échantillon'). They must therefore be used to perform statistical procedures. Each (unknown) variable must be associated with a confidence interval.

Recall on Confidence Interval: The *confidence interval* $[c_1, c_2]$ of the unknown parameter λ is defined through 2 statistical values C_1, C_2 so that it covers the unknown (true) value of λ with a given probability $1 - \alpha$, that is:

$$p(C_1 \leq \lambda \leq C_2) = 1 - \alpha.$$

The probability $1 - \alpha$, associated with this interval estimation, is called *a confidence level* ('niveau de confiance) or *confidence threshold* ('seuil de confiance'). The most commonly used values for $1 - \alpha$ are 0.9; 0.95; 0.99 and 0.999.

Each realization of the two statistics C_1, C_2 provides a numerical confidence interval $[c_1, c_2]$. So, the notion of *confidence level* is to be interpreted as follows: if we perform a large number of realizations of the two statistics (C_1, C_2), then the unknown value of the parameter will be covered by approximately $100(1 - \alpha)$ % of the obtained intervals $[c_1, c_2]$.

The length of a *confidence interval* decreases by:

- Increasing the size of the sample ('échantillon'),
- Reducing the dispersion of the considered random variable λ,
- Choosing a lower confidence level, for example by taking 0.9 instead of 0.95. In ARENA (when the number of replications is greater than 1), the *half width* ('demi-largeur') corresponds to a *confidence interval* with a threshold equal to 95%.

There are two types of systems:

- The *terminating* systems ('systèmes finis'), that is, having an end event that determines the end of the simulation, for example, a shop that opens and closes at regular intervals,
- The *non-terminating* systems, that is, having no end event of simulation, for example, a hospital where there is always at least one patient.

V.1 Analysis of Terminating Systems

Terminating systems are easier to analyze than the others. The number of repetitions/runs of the simulation model can be controlled allowing a different random number generator to be used for each repetition.

Two sources of observation data:

a) Individual observations in each repetition, for example, the processing time of each workpiece,
b) Means, standard deviations, maximums, minimums of observations in each repetition, for example, the average processing time of the workpieces.

If the random number generator changes from each repetition, observations of type *b*) of a set of repetitions can be considered such that:

- Observations are independent,
- Their means are normally distributed.

The latter property is due to the fact that the observations are derived from sums, or averages, of individual observations (*central limit theorem*). Hence, conventional statistical procedures can be applied for means. For minimums and maximums, certain statistical procedures still apply.
From the observations of type *b*), one can calculate in particular:

- Confidence intervals for mean, maximum and minimum,
- Confidence intervals around the difference between the means, maximums, and minimums of two different systems.

This comparison between two systems is useful to evaluate, for example, the difference between two sizings ('dimensionnement'), two scheduling rules, etc. (knowing that two systems are different if the confidence interval does not contain 0).

General procedure:

- Simulate a large number of repetitions (minimum 20) and collect each time the desired observations (average, maximum, minimum, etc.);
- Analyze system behavior based on the average value for each repetition:
    - Use of histogram,
    - Calculation of confidence interval.
- Determine the number of repetitions using the analysis of the results according to the desired precisions for the confidence interval. Use the formula $n_2 = n_1\ (h_1/h_2)^2$ where
    n_1 is the number of experiments already realized,
    n_2 is the total number of experiments,
    h_1 is the confidence interval already obtained,
    h_2 is the desired confidence interval.
- Simulate again: Either start all over again, or add the results of the new simulations to those of the first,
- Analyze: Confidence interval, histogram.

V.2 ANALYSIS OF NON-TERMINATING SYSTEMS

We are interested in the study of the *stationary performance* of a system in the sense that the *transient* regime/behavior is often favorable to system performance; this may be, for example, the case of an empty workshop at the simulation beginning. The steady state of the system corresponds to its behavior after a certain time and is independent of the starting state.
The goal is to calculate a confidence interval for the mean. Two problems may arise:

- No precise crossing point between the transient regime and the stationary regime,
- Correlation between observations.

Problem of the transient regime
Three methods exist to deal with the problem of the transient regime:

- Choose starting conditions that look like steady-state conditions, for example, by loading machines, by putting workpieces in queues,
- Make simulations long enough to make the effect of the transient regime insignificant,
- Exclude the values recorded during the transient regime for example, by using the sliding average filter ('filtre de la moyenne glissante') (arithmetic mean of *k* recent observations) to reduce the variability of the variable.

The latter method is commonly used. There are some rules for selecting the simulation part to truncate, but there is no completely satisfactory method. The most used is to evaluate (visually) the transient period using graphs (curves, histograms, moving averages).

Confidence intervals
Two methods are commonly used:

- Repetition of independent experiments as for finite systems (with the transient regime problem raised at each time),
- Long simulation and decomposition of data generated into subsets (*batches*).

The latter method consists of:

- Exclude the transient regime,
- Decompose the remaining observations by considering n *batches* of size m and without overlap ('chevauchement'),
- Replace each *batch* B_j $(j = 1, 2, \cdots, n)$ by X_j, mean of the m observations in B_j,
- Calculate the confidence interval from observations $X_j,\ j = 1, 2, \cdots, n$.

The conditions of the *central limit theorem* are assumed verified, and the calculation of the confidence interval justified (independence and normality of observations X_j).

Indications: $n = 10\ lag^*$,
m between 10 and 20.

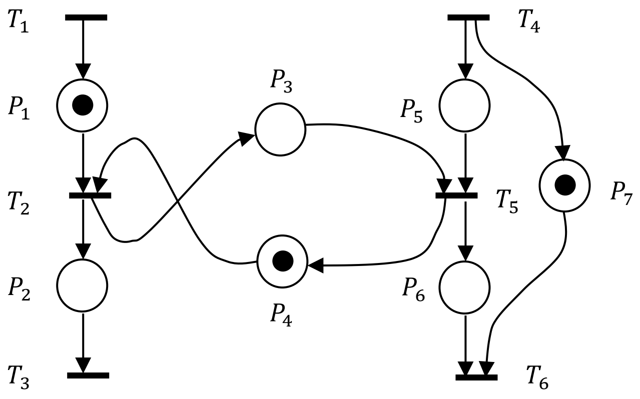
Correlogram $\rightarrow lag^*$: The largest number of observations for which the correlation is still significant.

This method (presented for non-time-dependent variables such as the number of finished workpieces) is obviously applicable for persistent (time-dependent) variables such as queue sizes. Simply define *batches* by regular time intervals instead of a fixed number of data.

VI BASIC NOTIONS ON PETRI NETS

VI.1 GENERALITIES

Definition (Petri net)
A *Petri net* is a graph with two types of nodes, namely *places* (represented by circles) and *transitions* (represented by bars). Places and transitions are connected by *arcs*. An arc is directed and runs from a place to a transition or vice versa (never between places or between transitions). See example in the following figure.



More formally, a Petri net can be defined by a 4-uplet $< P, T, Pre, Post >$ such that:

- $P = \{P_1,\ P_2, \cdots,\ P_n\}$ is a finite (non-empty) set of **places**,
- $T = \{T_1,\ T_2, \cdots,\ T_m\}$ is a finite (non-empty) set of **transitions**,
- Pre: $P \times\ T\ \rightarrow \{0,\ 1\}$ is an application such that $\boldsymbol{Pre}(\boldsymbol{P_i}, \boldsymbol{T_j})$ is the **weight** of the **arc running from the place $\boldsymbol{P_i}$ to the transition $\boldsymbol{T_j}$** (this weight is equal to 1 if the arc exists and to 0 elseif),
- $Post$: $P \times\ T\ \rightarrow \{0,\ 1\}$ is an application such that $\boldsymbol{Post}(\boldsymbol{P_i}, \boldsymbol{T_j})$ is the **weight** of the **arc running from the transition $\boldsymbol{T_j}$ to the place $\boldsymbol{P_i}$**.

P_i is said to be an *input place* of a transition T_j if an arc runs from place P_i to transition T_j.
P_i is said to be an *output place* of a transition T_j if an arc runs from transition T_j to place P_i.
In a similar way, a transition is said to be an *input* or an *output transition* of a place.

A transition without input place is a *source transition*, a transition without output place is a *sink transition*.

Marking
Places are *marked* with *tokens* (black dots). The tokens *circulate* in the places due to the *firing* of transitions following rule defined below. The *state* of the net at a time t is defined by the number of tokens contained in each *place* at time t. Circulation of tokens represents the dynamic evolution of the *state* of the net. The *initial marking* corresponds to the initial position of the tokens, that is, the one represented in the figure.

Firing of transitions, circulation of tokens in places
A transition is **enabled** if **all its input places** contain **at least one token**. Let us note that a *source transition* is always enabled.
An enabled transition can be **fired**. **Firing** of a transition consists of withdrawing one token from each of its *input places* and of adding one token to each of its *output places*.

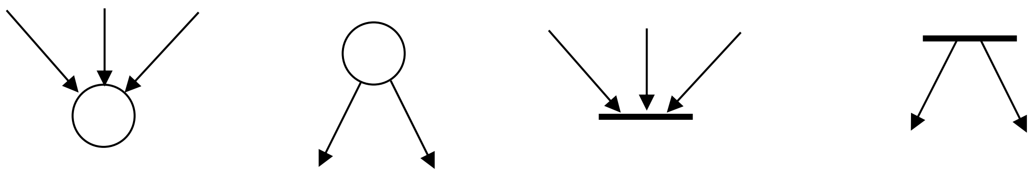
More formally, the firing of transition T_j consists of withdrawing $Pre(P_i, T_j)$ tokens in each of the places (P_i) and of adding $Post(P_k, T_j)$ tokens in each of the place (P_k) of the Petri net knowing that the firing of transition T_j is possible only if the marking of each place P_i of the Petri net is such that:

$$m(P_i) \geq \ Pre(P_i, T_j).$$

Exercise: Give a matrix expression of the applications Pre et $Post$ of the previous Petri net. Validate through some examples the correct functioning of the evolution of the marking equation: $M_k = M_l + (Post - Pre)\ T_{k\,l}$,
where $T_{k\,l}$ is the firing vector allowing an evolution of the marking vector from M_l to M_k.

Modeling of competition (OR *logic*) and synchronization (AND *logic*)

- *Competition/conflict for the supply of tokens in a place* due to the convergence of arcs on the place (see figure *a* below).
- *Competition/conflict to the consumption of tokens from a place* due to the divergence of arcs from the place (see figure *b* below). This *structural* conflict must be arbitrated by some priority rule when the conflict is *effective*, that is, when several *output transitions* of the place could be fired. The behavior of the net is not fully specified when a conflict is *effective*.
- *Synchronization in the consumption of tokens of several places* due to the convergence of several arcs on a transition (see figure *c* below).
- *Synchronization in the provision of tokens of several places* due to the divergence of arcs from a transition (see figure *d* below).

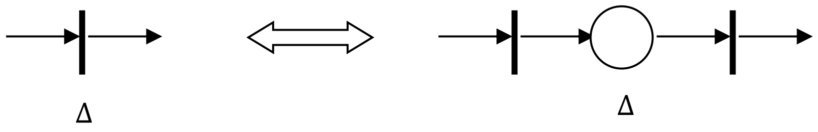


Association of time with places and/or with transitions

Petri nets are said *timed* (‘RdP *temporisés*’) when time is associated with places and/or transitions. The activation of a transition can be interpreted as the execution of a task which is a reason for associate a time with transitions. On the other hand, if a place is perceived as a place where a resource must stay while waiting to continue his progression in the Petri net, there may be a *minimum* duration of sojourn to respect: For example consider the sojourn of a biscuit in an oven (the resource) to reach a desired temperature.
So, we are tempted to put both:

- a time associated with transitions: The time during which a token located in each *input place* of the fired transition is ‘reserved’ for this transition (before disappearing), and beyond which a token appears in each *output place*,
- a *minimum* time of token sojourn in places: The time during which any token that has just been introduced in a place cannot be used to fire an *output transition*.

In fact, there is no loss of generality in associating time *only* on transitions, or *only* on places. The following figure shows the transformation of a Δ timed transition into 2 instantaneous transitions separated by a Δ timed place.



VI.2 EVENT GRAPHS

Constraints and modeling capabilities

Event graphs are a subclass of Petri nets for which every place has exactly one *input* and one *output transition* (the situations represented in previous figures *c* and *d* are possible contrary to the ones represented in figures *a* and *b*). So, event graphs can model *synchronization* phenomena but not *competition/conflict* ("*concurrence*").

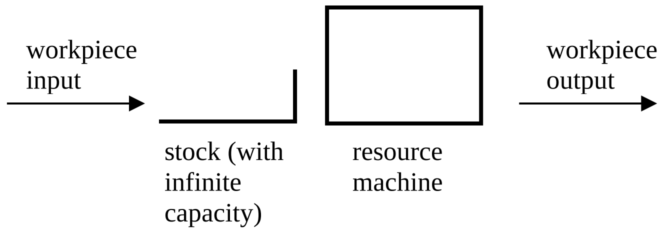
Dually, *state graphs* are a subclass of Petri nets for which every transition has exactly one *input* and one *output place* (the situations represented in figures *a* and *b* are possible contrary to the ones represented in figures *c* and *d*). So, such graphs can model *competition/conflict* phenomena but not *synchronization*.

A fundamental property of event graphs

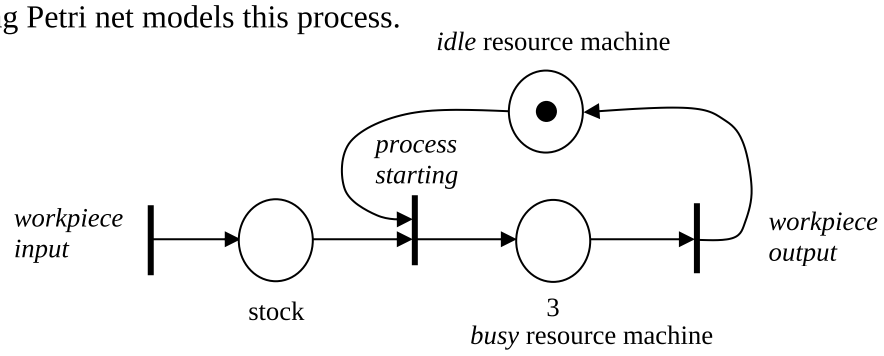
The total number of tokens along any circuit of an event graph remains constant (which is not always verified in a Petri net).

VI.3 EXAMPLES

Consider the machine depicted in the following figure. Each workpiece that arrives at the stock input is either processed immediately by the resource machine or put on hold ('en attente') in the stock until the availability of the resource machine. The processing time of the resource machine is 3 units of time. After treatment, each workpiece comes out of the machine.

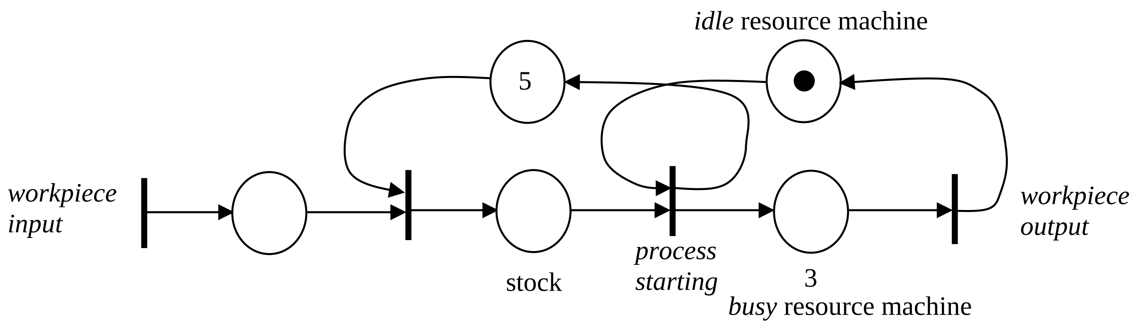


The following Petri net models this process.

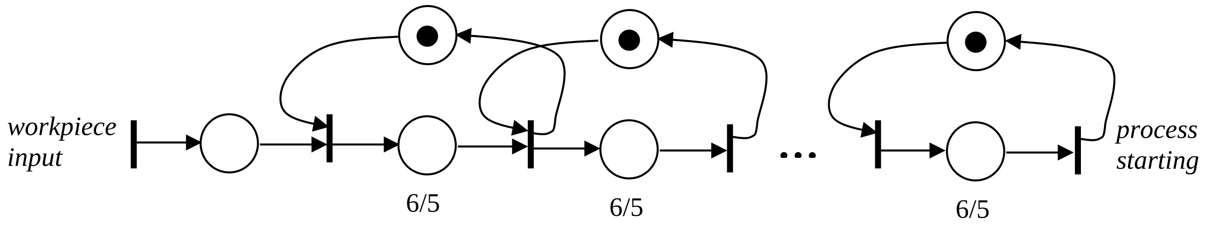


Modifications

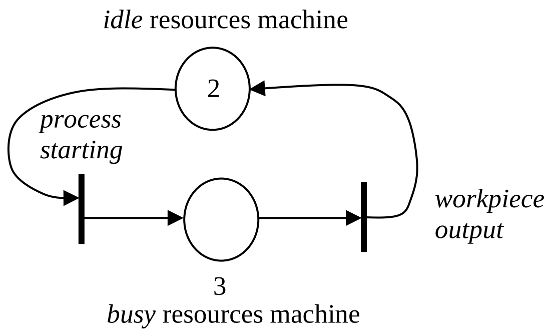
a) The following Petri net limits the storage capacity of the stock to 5 workpieces.



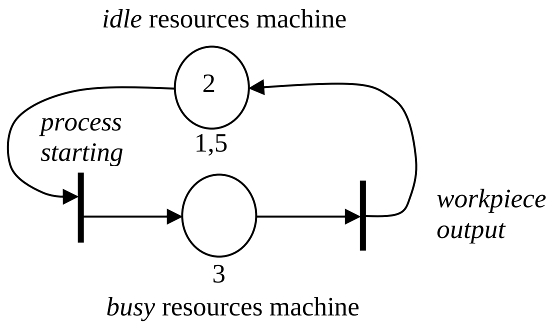
b) The stock is replaced by a conveyor corresponding to a queue composed of 5 compartments (with *First-In, First-Out* rule). The travel time of the conveyor is equal to 6 units of time. The process is represented by the following Petri net.



c) The machine has a processing capacity of 2 which allows the processing of 2 workpieces simultaneously. The process is represented by the following Petri net.



d) The machine has a *setup* time equal to 1,5 units of time. The process is represented by the following Petri net.



VI.4 OTHER CLASSES OF PETRI NETS

Many extensions of Petri nets are possible, let us briefly present some of them.

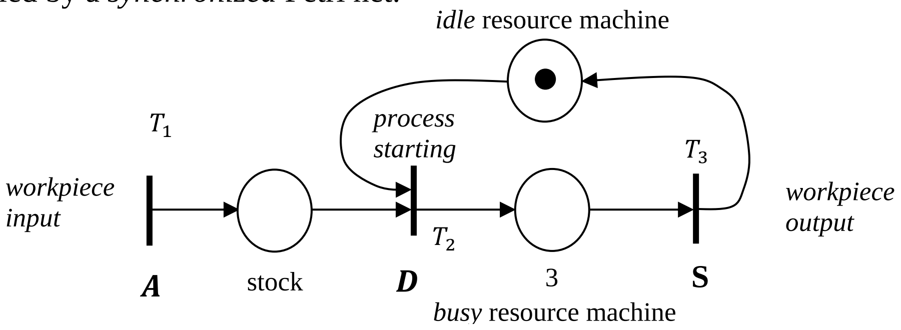
***Time* Petri nets ('RdP *temporels*')**

Time Petri nets allow the analysis of time-constrained systems. An interval (rather than a simple scalar value) is associated with places and/or transitions. Associating interval $[a, b]$ with a place means that a token present in that place will have to stay at least a units of time. It can contribute to the enabling of an *output transition* if its residence time is less than b units of time, beyond this time the token 'dies' and therefore will no longer contribute to the enabling of an *output transition*. Such nets are called P-*time* Petri nets knowing that Petri nets with interval associated with transitions (called T-*time* Petri nets) have a different functioning mode.

***Synchronized* Petri nets**

A set of *external* events is associated with Petri net; these events allow the firing of certain transitions. Such nets are called *synchronized* Petri nets.

Consider for example the Petri net modeling the machine described in VI.3. Set of events $\{A, D, S\}$ is associated with the Petri net where A designates the event '*Workpiece arrival'*, D the event '*Service start'*, and S the event '*Workpiece exit'*. The following figure shows the system modeled by a *synchronized* Petri net.

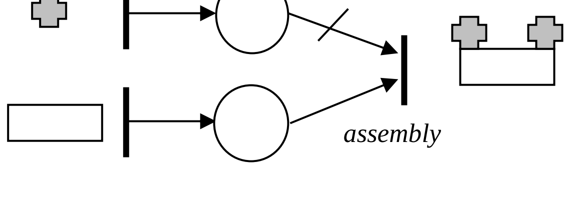


Firing of the *source transition* T_1 depends on the occurrence of the event A.
Firing of the transition T_2 is linked to:
- its validation, materialized by the presence of one token in its two *input places*,
- the occurrence of the event D.

Firing of the transition T_3 depends on the occurrence of the event S and the presence of one token during 3 units of time in its *input place*.

***Generalized* Petri nets**

A *generalized* Petri net is a Petri net in which weights associated with arcs are strictly positive integers (not only equal to 0 or 1 as previously) knowing that (as previously) the weight of an arc equal to 1 is not explicitly specified.
Let be an arc running from a transition T_j to a place P_i with a weight equal to p, then the firing of this transition means that p tokens will be added to the place P_i.
Let be an arc running from a place P_i to a transition T_j with a weight equal to p, then the transition T_j is enabled only if the place P_i contains at least p tokens. The firing of this transition means that p tokens will be removed from the place P_i.



VII SIMULATION USING ARENA

SIMAN was a simulation software designed in 1982 by C.D. Pedgen. ARENA[4] represents the graphical version of SIMAN and was acquired by Rockwell Automation in 2000, it was based on the use of blocks provided by libraries, also called *templates*. The construction of a *block diagram/flowchart* allows the description of the logic of the process to modelize, it corresponds to the *static component* of the simulation model. The *dynamic behavior* of the simulation model is described by the flows of *'entities'* moving through the *block diagram*.

VII.1 BASIC NOTIONS

Entity: An *entity* is an object of interest in the system whose movement in the model may cause the occurrence of events, similarly to the movement of a token in a Petri net. An entity flows through the different blocks of the *block diagram* by potentially using some ***resources*** of the system. It usually corresponds to a concrete object, for example, a workpiece in a workshop, a patient in a hospital or a customer in a restaurant.

Attribut: An *attribute* is a variable *individually* associated to an entity, so its value is specific to the entity. For example, an entity considered as a product may have attributes *serial number*, *weight*, and *price*. All products have these attributes, but they do not necessarily have the same values for each attribute. As a second example, consider a workpiece circulating in a workshop as an entity with 2 attributes: *type_of_workpiece* to designate its type (*A* or *B*) and *index_of_priority* to designate its priority (*low* or *high*) of passage on the machines that will process it.

Resource: A *resource* is a limited quantity of *identical resource units* used by entities. Each *resource unit* has a *busy* or *idle* status (*busy* when the resource is seized, *idle* when it is released). An entity must wait in a ***queue*** if it attempts to seize a resource while all its resource units are seized. For example, consider a machine as a resource with a capacity of 3 *resource units* which allows the processing of 3 workpieces simultaneously or an airport counter with 2 employees (corresponding to 2 identical resource units) which allows to satisfy two customers simultaneously.

Queue: A *queue* is a location that holds entities when their movement is constrained within the *block diagram*.

Variable: The value of a *variable* is available for all the blocks of the model and can be shared between all the entities within the model. For example, *TNOW* is a variable predefined in ARENA that refers to the date in which the simulation is located, it is the current time (updated with each advance in the schedule of events) that elapses during a *run* of the model.

The aim is to allow the evolution of *each entity* within the *block diagram* from one block to the following one, from its creation to its destruction. The scheduling over time ('ordonnancement dans le temps') of the different events related to the evolution of the entities in the blocks (composing the *block diagram*) is done through a *schedule* ('échéancier').

[4]Electronic documentation is provided with the ARENA software through various files (*Getting Started with Arena.pdf, OptQuest for Arena User's Guide.pdf, Packaging Template User's Guide.pdf, Variables and Functions Guide.pdf*) available in the \Rockwell Software\\Arena\\Online Books directory.

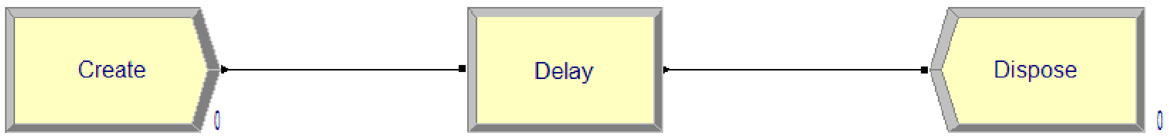
A simulation model is composed of two parts:

- The first part is the '*model part*' of the simulation and is built in the *model window*, it describes the logic of the process to modelize through the assembly (in series, in parallel or in feedback) of different blocks to form a block diagram. The *dynamic* behavior of the simulation model is represented by moving entities from block to block through the *block diagram*, thus activating the *function/service* associated with each of these blocks. A *function/service* can act on:
  - the *attribute* of the entity (the one activating the *function/service*). For example, the entry of an entity (considered as a workpiece) in an ***Assign*** block allows to fix a value to its *serial number* attribute,
  - a *variable* of the simulation model. For example, the entry of an entity in a ***Delay*** block causes a pure delay of the entity which will affect the *TNOW* variable.

- The second part is the '*experiment part*' and deals with the *experimental conditions* of the simulation. Blocks used, called *data* blocks, concern for example the replication length, the number of replications, the characteristics of resources and queues used in the *block diagram*. Blocks are edited *via* spreadsheet interface ('feuille de calcul') to display their model data.

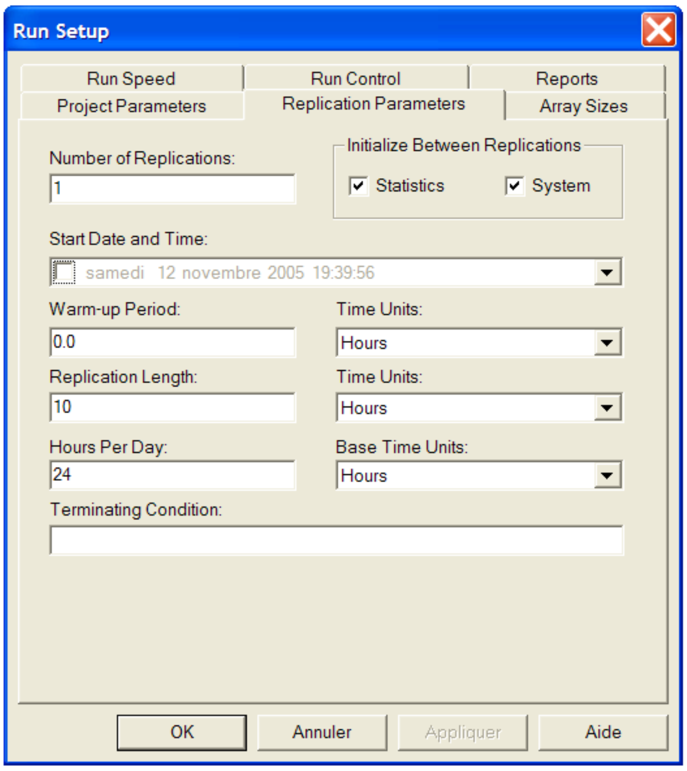
N.B.: Only blocks used to construct the *block diagram* can be activated by entities. Entities never pass through the blocks belonging to the *experiment part* (note that these blocks are graphically represented by a kind of spreadsheet to distinguish them from those used to build the block diagram).

Consider a simple conveyor with a transport time equal to 3 units of time represented by the simulation model (*Example.doe*) described as follows:



The ***Create*** block (from the *Discrete Processing template*) is such that an entity is created every 2 units of time from time 0.
The ***Delay*** block (from the *Discrete Processing template*) forces an entity to stay 3 units of time in the block.
The ***Dispose*** block (from the *Discrete Processing template*) destroys all entities entering the block.

We can fix through the menu *Run/Setup…/Replication Parameters*:
- the number of replications (*Number of Replications* field),
- the length of the replication (*Replication Length* field).



Number of Replications: 1

Initialize Between Replications: [✓] Statistics [✓] System

Start Date and Time: [ ] samedi 12 novembre 2005 19:39:56

Warm-up Period: 0.0 Time Units: Hours

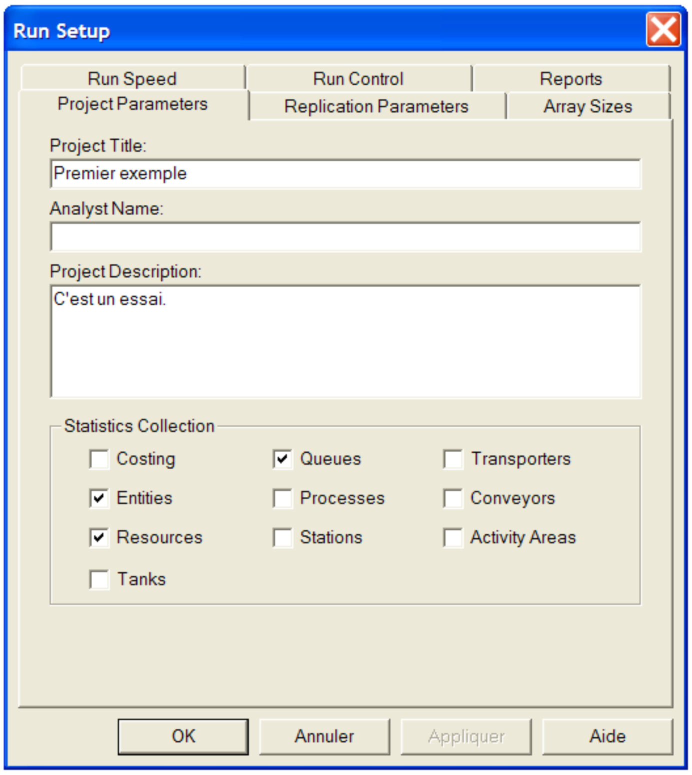
Replication Length: 10 Time Units: Hours

Hours Per Day: 24 Base Time Units: Hours

Terminating Condition:

We can fix through the menu *Run/Setup…/Project Parameters*:

- a title to the project (*Project Title* field),
- the name of the analyst (*Analyst Name* field),
- a comment about the project (*Project Description* field).



Project Title: Premier exemple

Analyst Name:

Project Description: C'est un essai.

Statistics Collection

[ ] Costing	[✓] Queues	[ ] Transporters
[✓] Entities	[ ] Processes	[ ] Conveyors
[✓] Resources	[ ] Stations	[ ] Activity Areas
[ ] Tanks		

The two *txt* files relative to *model part* and *experiment part* are accessible through the menu *Run/SIMAN/View*, see below an extract of these files.

- **file *Example.mod* corresponding to the *model part* of file *Example.doe*:**

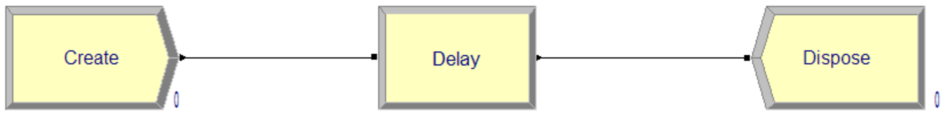
```
; Model statements for module:  Create
2$          CREATE,          1,HoursToBaseTime(0.0),Entity 1:HoursToBaseTime(2):NEXT(0$);
;
; Model statements for module:  Delay
0$          DELAY:           3,,Other:NEXT(1$);
;
; Model statements for module:  Dispose
1$          DISPOSE:         Yes;
```

0$, 1$, 2$ are labels.

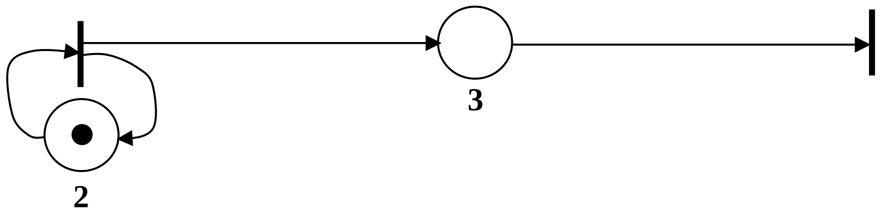
- **file *Example.exp* corresponding to the *experiment part* of file *Example.doe*:**

```
PROJECT,            "Premier exemple","",,,No,Yes,Yes,Yes,No,No,No,No,No,No;
REPLICATE,          1,,HoursToBaseTime(10),Yes,Yes,,,,24,Hours,No,No,,,Yes;
```

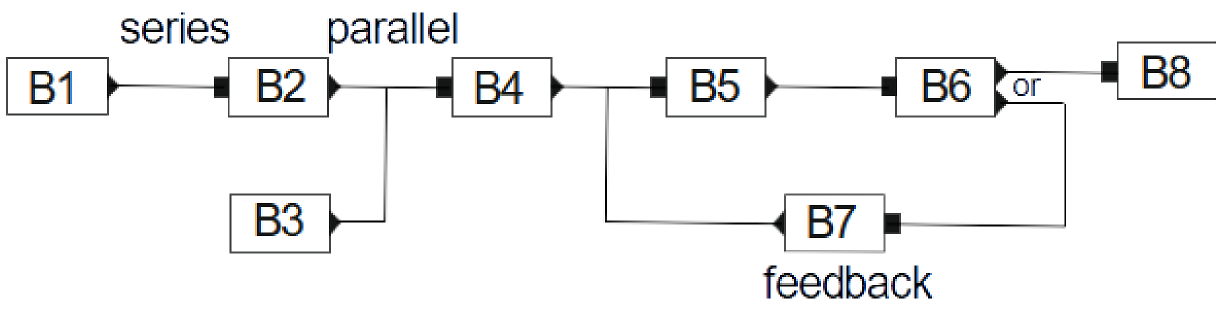
The Petri net corresponding to the *model part*, that is,

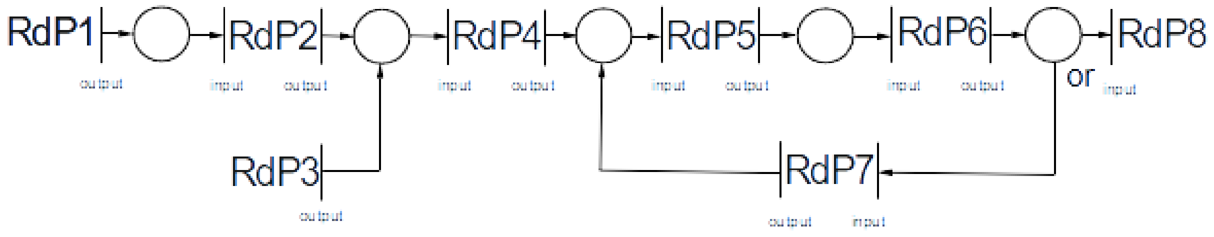


is described in the following figure:



Blocks of the *block diagram* are joined in series, in parallel, or in feedback by *connectors*. A *connector* transfers an entity in zero time from the output point of a block (graphically represented by a small arrow) to the input point of the next block (represented by a small rectangle). An example of such block assemblies is given in the following figure with its equivalent Petri net.





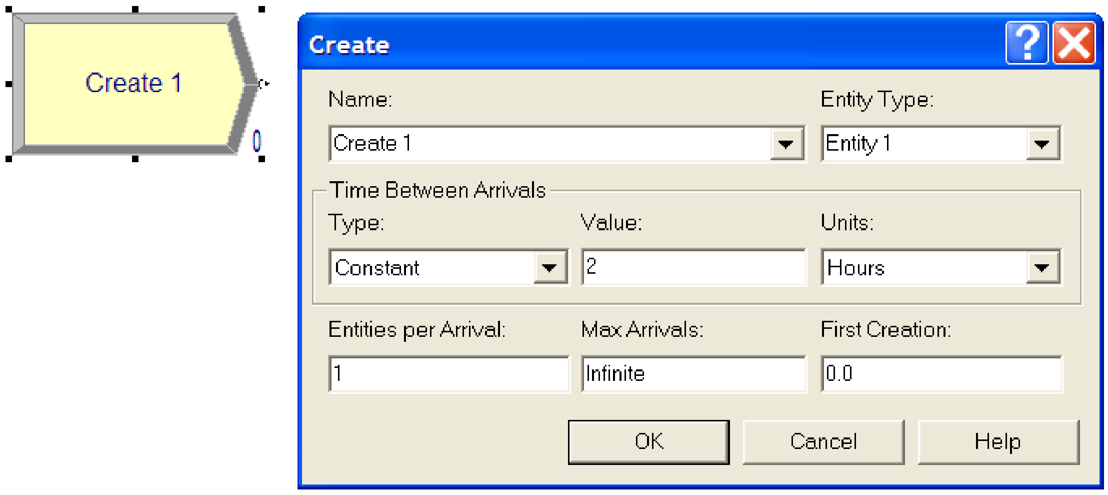
VII.2 BLOCKS TO BUILT MODEL

a) ***Create*** (from the template *Discrete Processing*, belonging to the *model part*): A ***Create*** block is used to enter entities in the model.

For example, the ***Create*** block shown in the following figure is entitled *Create* 1 (*Create* 1 → *Name* field). Entities enter the model sequentially according to a pattern specified in the *Time Between Arrivals* frame:

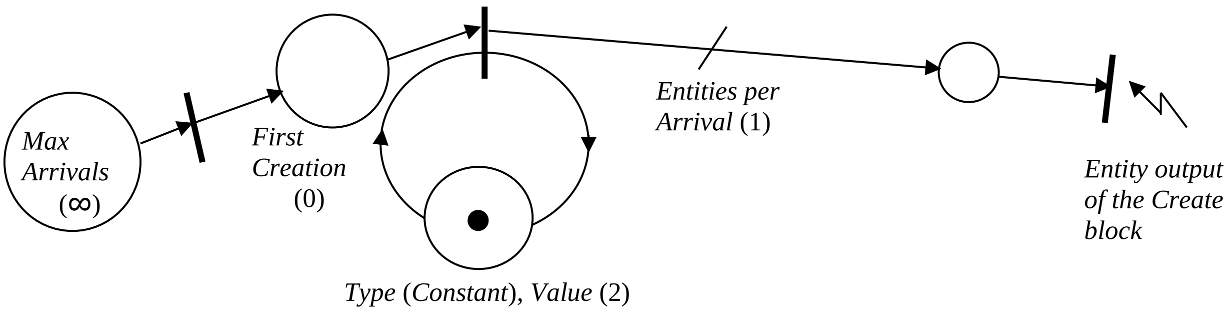
- the interarrival time, that is, the delay between two successive batch creations of entities ('création de lots d'entités'), is equal to 2 (*Constant* → *Type* field and 2 → *Value* field) (it be would be specified as an exponential distribution with a mean of 1 if *Expression* → *Type* field and *EXPO*(1) → *Value* field);
- the batch size to specify the number of entities in each batch creation is equal to 1 (1 → *Entities per Arrival* field);
- the total number of batches to be created is infinite (*Infinite* → *Max Arrivals* field);
- the date of creation of the first batch is equal to 0 (0 → *First Creation* field).

In other words, the values considered in the figure are such that one entity is created an infinite number of times every 2 units of time from time 0.



Name: Create 1
Entity Type: Entity 1
Time Between Arrivals
Type: Constant
Value: 2
Units: Hours
Entities per Arrival: 1
Max Arrivals: Infinite
First Creation: 0.0

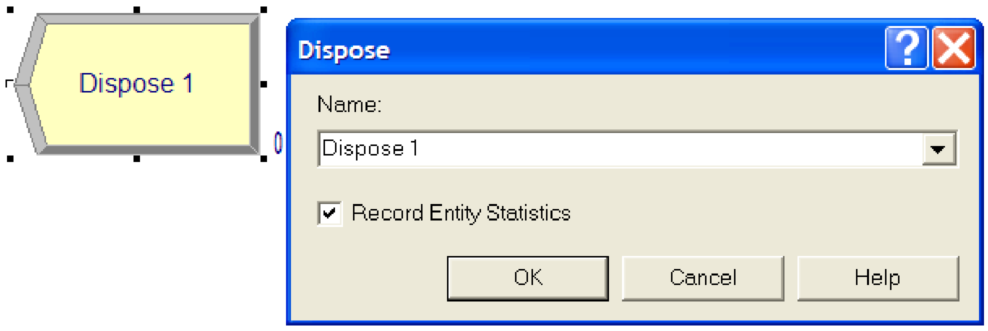
The following Petri net describes the behavior of the ***Create*** block.


Max Arrivals (∞)
First Creation (0)
Entities per Arrival (1)
Entity output of the Create block
Type (*Constant*), *Value* (2)

The number of firings of the output transition of this Petri net at a time *t* corresponds to the number of entities output from the *Create* block at the same time *t*. Note that a *Create* block has no input.

b) *Dispose* (from the template *Discrete Processing*, belonging to the *model part*): A ***Dispose*** block is used to remove entities from the model.

The ***Dispose*** block shown in the following figure is entitled *Dispose* 1 (*Dispose* 1 → *Name* field), an entity entering this block is immediately removed from the model.

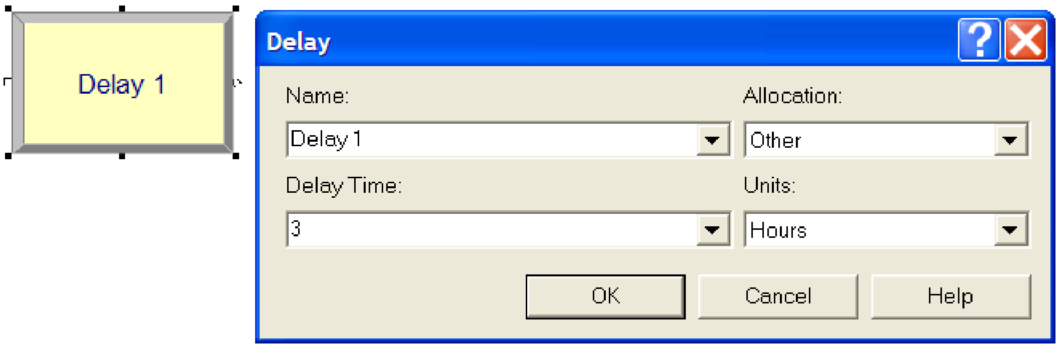


Name: Dispose 1
[✓] Record Entity Statistics

In Petri net terms, this block, which has no output, is equivalent to a *sink transition*, that is, a transition without *output place*.

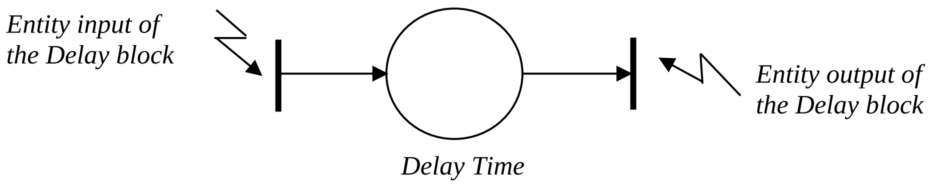
c) *Delay* (from the template *Discrete Processing*, belonging to the *model part*): Once the required number of resource units has been assigned to an entity, it generally engages in time-consuming activities, such activities can be modelled by using a ***Delay*** block.

The ***Delay*** block shown in the following figure is entitled *Delay* 1 (*Delay* 1 → *Name* field). When an entity enters this block, it remains there unconditionally for a duration equal to 3 units of time (3 → *Delay Time* field).



Name: Delay 1
Allocation: Other
Delay Time: 3
Units: Hours

The following Petri net describes the behavior of the ***Delay*** block.


Entity input of the Delay block
Entity output of the Delay block
Delay Time

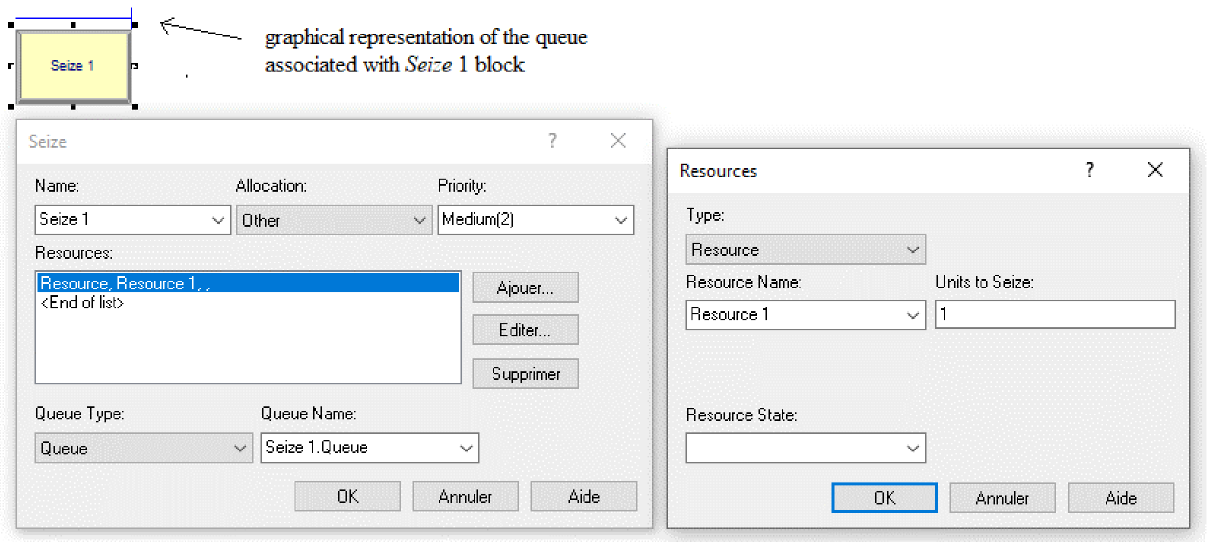
The number of tokens in the place at a time *t* corresponds to the number of entities in the ***Delay*** block at the same time *t*.

d) *Seize* (from the template *Discrete Processing*, belonging to the *model part*): A ***Seize*** block allows the allocation of *idle resource units* to entities entering this block. An entity waits in the internal queue of the block until it can seize the required number of *idle resource units*. When they are seized, their status becomes *busy* (they cannot be allocated for other entities as long as their status does not change) and the entity exits the ***Seize*** block.

For example, the block shown in the following figure is entitled *Seize* 1 (*Seize* 1 → *Name* field). To simplify consider that only one resource is required by an entity present in the block, then:

- the name of the resource, that is *Resource* 1, is specified in the *Resource Name* field (*Resource* 1 → *Resource Name* field). Note that the addition of another resource would produce an extra line in the list *Resources*;
- the number of *idle resource units* (of *Resource* 1) required by an entity, that is 1, is specified in the *Units to Seize* field (1 → *Units to Seize* field).

The entities waiting for the required number of *idle resource units* are put in a queue associated with the ***Seize*** block. Its name, that is *Seize* 1.*Queue* (name by default), is indicated in the *Queue Name* field (*Seize* 1.*Queue* → *Queue Name* field).


Seize 1
graphical representation of the queue associated with *Seize* 1 block

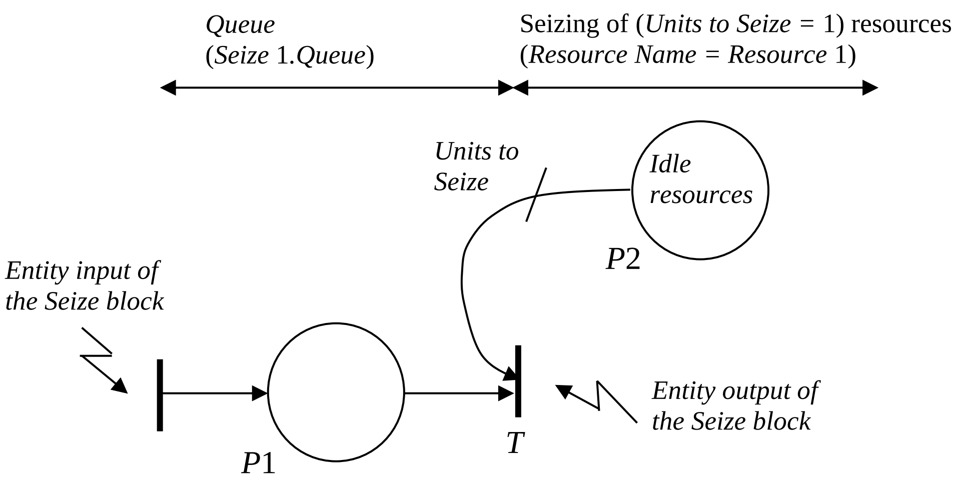
Seize

Name: Seize 1
Allocation: Other
Priority: Medium(2)
Resources:
Resource, Resource 1, ,
<End of list>
Ajouer...
Editer...
Supprimer
Queue Type: Queue
Queue Name: Seize 1.Queue
OK Annuler Aide

Resources

Type: Resource
Resource Name: Resource 1
Units to Seize: 1
Resource State:
OK Annuler Aide

The following Petri net describes the behavior of the ***Seize*** block using the resource *Resource* 1.


Queue
(*Seize* 1.*Queue*)
Seizing of (*Units to Seize* = 1) resources
(*Resource Name* = *Resource* 1)
Units to Seize
Idle resources
*P*2
Entity input of the Seize block
*P*1
T
Entity output of the Seize block

The transition *T* is *enabled* if:

- at least one token is in the place *P*1 which means that at least one entity is in the queue associated with the block ***Seize***,
- at least *Units to Seize* tokens are in the place *P*2 which means that at least *Units to Seize resource units* (of *Resource* 1) are idle.

Firing of the transition *T* both removes one token in the place *P*1 and removes *Units to Seize* tokens in the place *P*2 which means that *Units to Seize* resource units (of *Resource* 1) are seized when the entity exits the block.

The number of tokens in the place *P*1 at a time *t* corresponds to the number of entities waiting in the queue associated with the block ***Seize*** at the same time *t*.

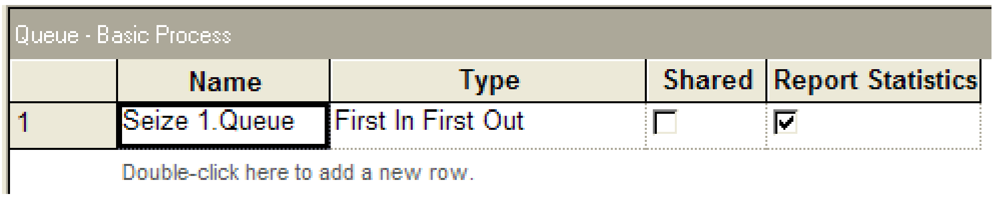
1. Each queue used in the simulation model is defined in a ***Queue*** block (from template *Data Definition*, belonging to the *experiment part*).

   For example, see the block *Queue* described in the following figure in which:
   - the *Name* field gives a name to the queue, that is *Seize* 1.*Queue* (*Seize* 1.*Queue* → *Name* field),

- the *Type* field indicates the ranking ('classement') rule establishing the relative position of each waiting entity in the *queue*, that is *First In, First Out* (*First In, First Out* → *Type* field). This option (FIFO) is the rule by default, the other options are: *Last In, First Out* (LIFO), *Lowest Attribute Value* (LAV), *Highest Attribute Value* (HAV). These last two options allow to prioritize the entities present in the queue by considering the *lowest* (or the *highest*) value of an attribute specified in the *Attribute Name* field attached to the used rule.

The ***Queue*** block allows the definition of several queues in a same model.

Queue



Queue - Basic Process

	Name	Type	Shared	Report Statistics
1	Seize 1.Queue	First In First Out	☐	☑

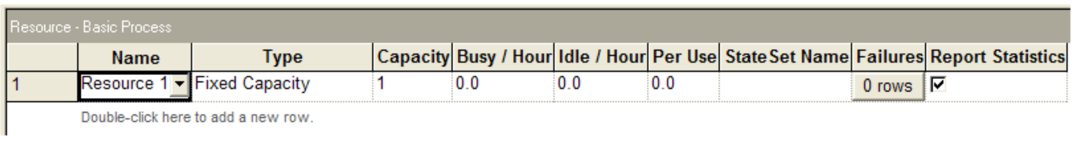
Double-click here to add a new row.

2. The name and the capacity (that is, the number of *resource units*) of each resource are defined in a ***Resource*** block (from template *Data Definition*, belonging to the *experiment part*).

   For example, see the block *Resource* described in the following figure in which:
   - the *Name* field gives the resource name, that is *Resource* 1 (*Resource* 1 → *Name* field),
   - the *Capacity* field gives the number of *resource units*, that is 1 (1 → *Capacity* field).

   The ***Resource*** block allows to define several resources in a same model.

Resource



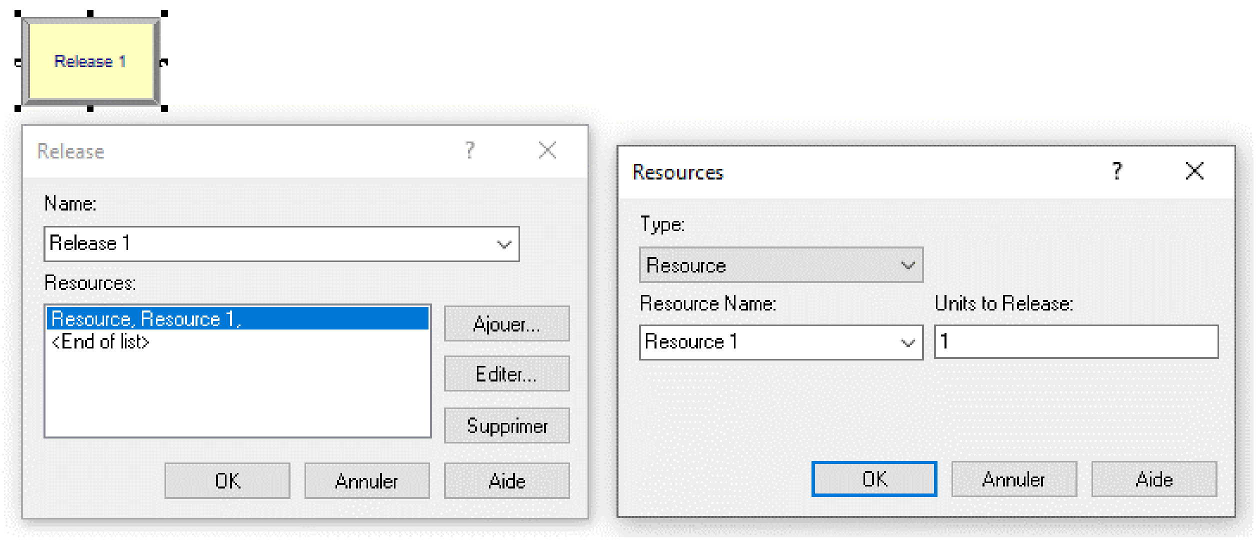
Resource - Basic Process

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics
1	Resource 1	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑

Double-click here to add a new row.

e) *Release* (from template *Discrete Processing*, belonging to the *model part*): When an activity requiring *resource units* is completed, the entity that possesses the *resource units* usually releases them so that they can be allocated to other entities. An entity in a ***Release*** block releases a quantity of *resource units* specified in the *Units to Release* field. When the *resource units* are released, their status change to *idle* (so they can be allocated for other entities) and the entity exits the ***Release*** block.

For example, the block shown in the following figure is entitled *Release* 1 (*Release* 1 → *Name* field). To simplify only one resource is released by an entity present in the block, then:
- the name of the released resource is specified in the *Resource Name* field, that is *Resource* 1 (*Resource* 1 → *Resource Name* field). Note that the addition of another resource would produce an extra line in the *Resources* list;
- the number of *resource units* (of *Resource* 1) released by an entity is specified in the *Units to Release* field, that is 1 (1 → *Units to Release* field).


Release 1

Release

Name:

Release 1

Resources:

Resource, Resource 1,
<End of list>

Resources

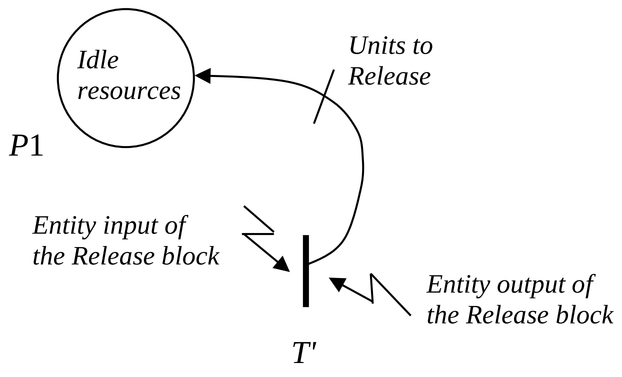
Type:

Resource

Resource Name: Resource 1

Units to Release: 1

The following Petri net describes the behavior of the ***Release*** block using the *Resource* 1 resource.


Idle resources
P1
Units to Release
Entity input of the Release block
Entity output of the Release block
T'

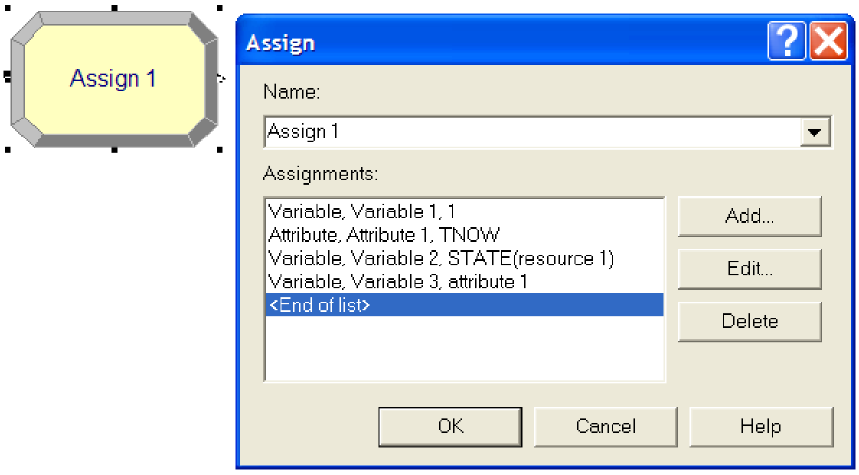
The firing of the transition *T'* adds *Units to Release* tokens in the place *P*1 which means that *Units to Release* resource units (of *Resource* 1) are released when the entity exits the *Release* block.

f) ***Assign*** (from template *Discrete Processing*, belonging to the *model part*): An ***Assign*** block is used to assign a *value* to an *attribute* or a *variable* during model execution. When an entity enters in an ***Assign*** block, the logical, or mathematical, expression specified in the *New Value* field is assigned to an *attribute* if *Attribute* → *Type* field or to a *variable* if *Variable* → *Type* field. Note that the *value* can be any valid ARENA expression, moreover multiple assignments can be made at a single ***Assign*** block.

In the following figure, the block untitled *Assign* 1 (*Assign* 1 → *Name* field) sequentially assigns the values:

- 1 to the *Variable* 1 variable,
- TNOW to the *Attribute* 1 attribute,
- *STATE*(*resource* 1) to the *Variable* 2 variable knowing that *STATE*(*resource* 1) is an ARENA expression that gives the current state of the mentioned resource (that is *resource* 1). The possible values of this state are: -1(=*Idle*); -2(=*Busy*); -3(=*Inactive*); -4=(*Failed*),
- *Attribute* 1 to the *Variable* 3 variable.

This example is provided in **\Examples\Assign\Assign.doe**.


Assign 1

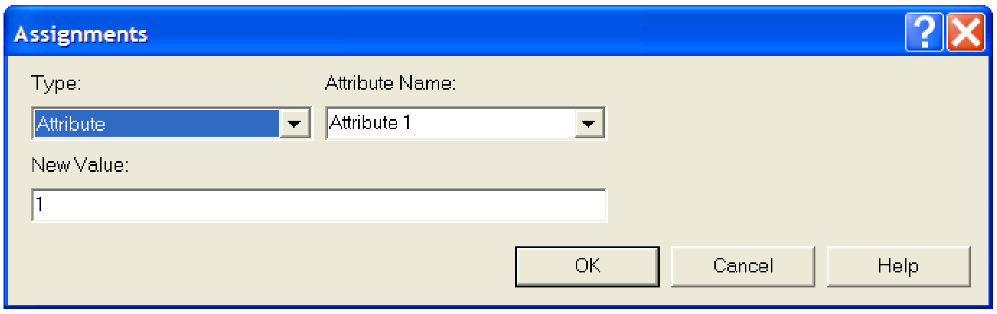
Assign

Name:

Assign 1

Assignments:

Variable, Variable 1, 1
Attribute, Attribute 1, TNOW
Variable, Variable 2, STATE(resource 1)
Variable, Variable 3, attribute 1
<End of list>

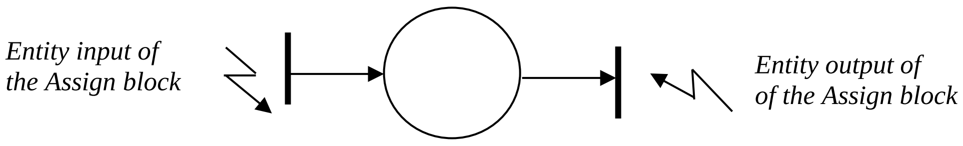

Assignments

Type: Attribute

Attribute Name: Attribute 1

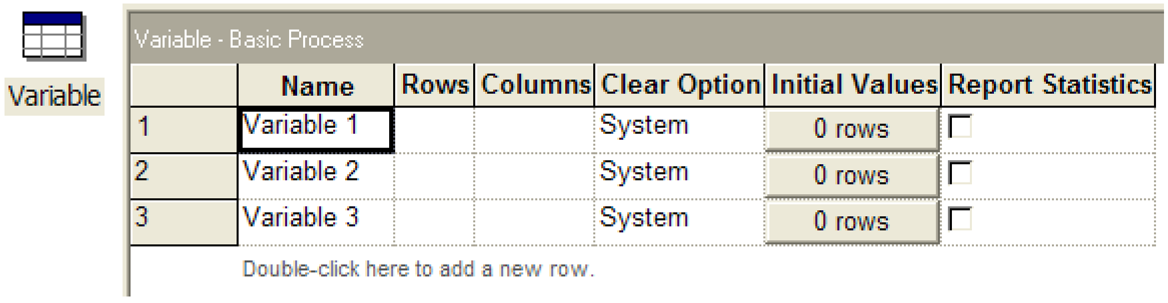
New Value: 1

The following Petri net describes the behavior of the ***Assign*** block *Assign* 1.


Entity input of the Assign block
Entity output of the Assign block

$$\begin{aligned} &\mathit{Variable}\ 1 = 1 \\ &\mathit{Attribute}\ 1 = \mathrm{TNOW} \\ &\mathit{Variable}\ 2 = \mathrm{STATE}(\mathit{resource}\ 1) \\ &\mathit{Variable}\ 3 = \mathit{Attribute}\ 1 \end{aligned}$$

A ***Variable block*** (from template *Data Definition*, belonging to the *experiment part*) is used to declare variables, that is *Variable* 1, *Variable* 2, and *Variable* 3 in the previous example as described in the following figure.



Variable - Basic Process

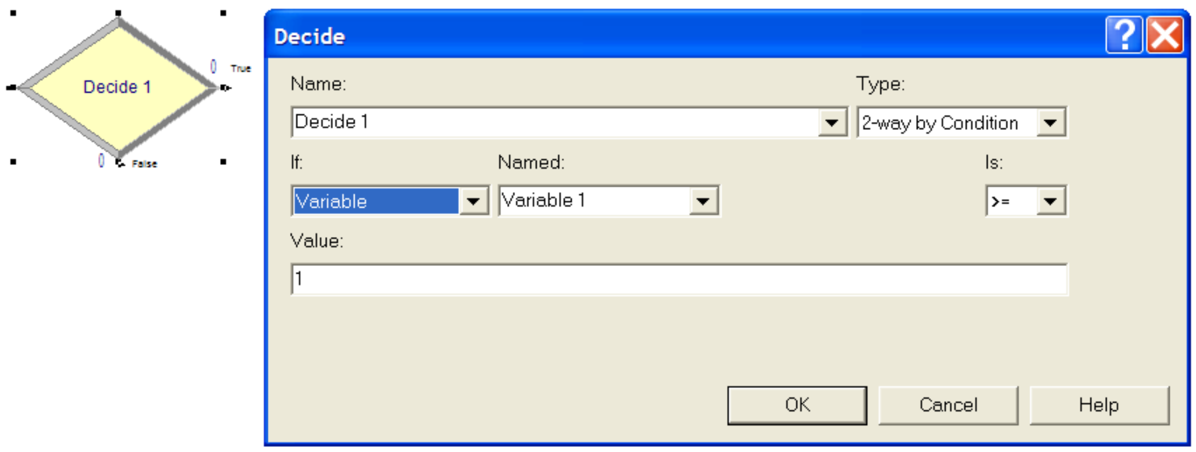
	Name	Rows	Columns	Clear Option	Initial Values	Report Statistics
1	Variable 1			System	0 rows	☐
2	Variable 2			System	0 rows	☐
3	Variable 3			System	0 rows	☐

Double-click here to add a new row.

g) *Decide* (from template *Decisions*, belonging to the *model part*): An entity that enters in the (single) input port of a ***Decide*** block can take different paths out of the block. The path can be chosen based on a condition or set of conditions or based on a probability distribution according to the *Type* field.

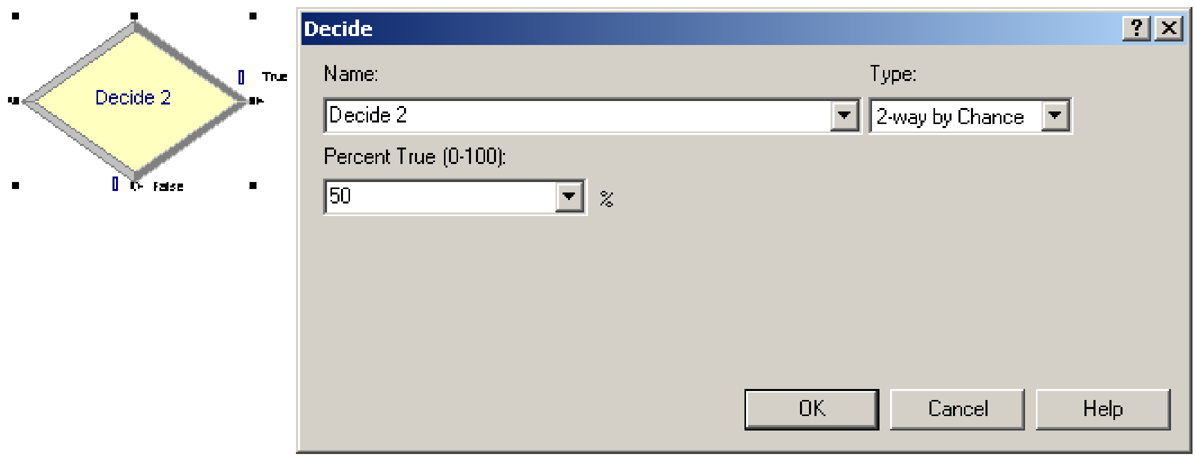
When an entity enters in a ***Decide*** block, each branching condition is tested sequentially. The branch selected by an entity is the first branch for which the *branching condition* is satisfied. If no branch is satisfied, the entity is pointed to the *Else* branch.

For example, a ***Decide*** block, entitled *Decide* 1 (*Decide* 1 → *Name* field), is described in the following figure. The branching condition towards the 2 possible outputs depends on the condition: *If Variable* 1 >= 1 (with a result *True* or *False*).



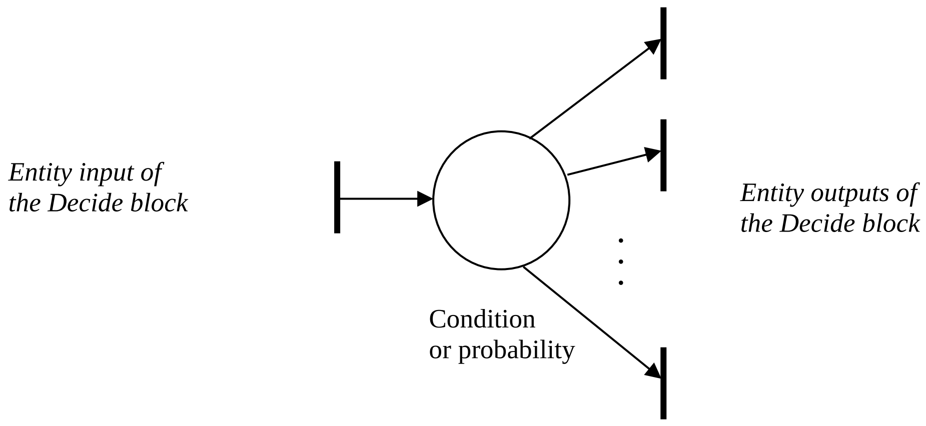
Name: Decide 1
Type: 2-way by Condition
If: Variable
Named: Variable 1
Is: >=
Value: 1

The branching condition used in the *Decide* 2 block, described in the following figure, is a probability law, there are 2 outputs with a probability equal to 0.5 for each of one.



Name: Decide 2
Type: 2-way by Chance
Percent True (0-100): 50 %

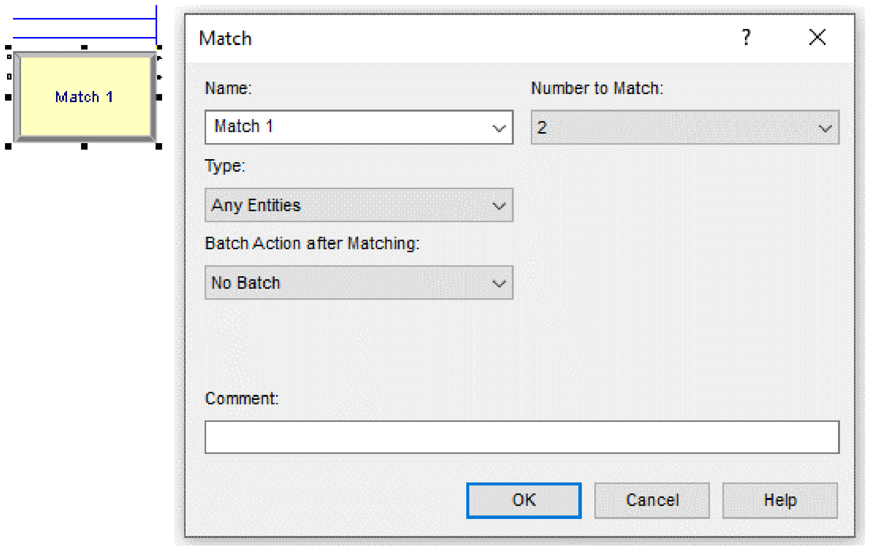
The following Petri net describes the behavior of the ***Decide*** block.


Entity input of the Decide block
Condition or probability
Entity outputs of the Decide block

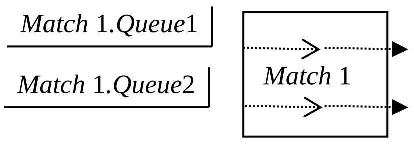
Note that all outputs of a ***Decide*** block must be connected to a block (possibly a ***Dispose*** block if the branch is not used).

h) ***Match*** (from template *Grouping*, belonging to the *model part*): A ***Match*** block is used to match/synchronize entities each located in different *matching queues* associated with the block. To each of these *matching queues* corresponds an output which allows each entity contained in a queue to exit by the corresponding output. When all these queues contain at least one entity, these entities are released simultaneously and routed to their corresponding outputs.

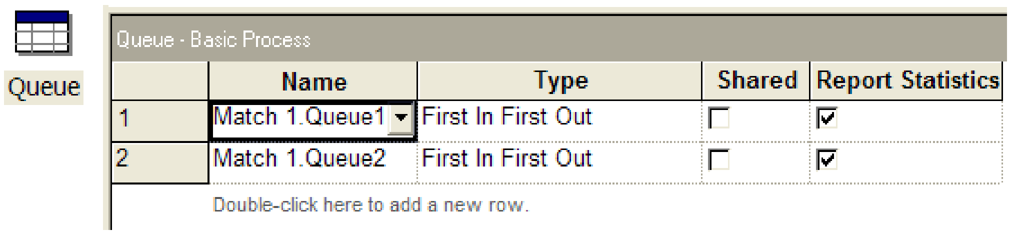
For example, the *Match* 1 block (*Match* 1 → *Name* field) described in the following figure executes a match between its two *matching queues*, that is, *Match* 1.*Queue*1 and *Match* 1.*Queue*2, represented by two blue lines located above the block. A synchronization occurs when at least one entity is present in each of these two queues (*Any Entities* → *Type* field) (note that the synchronization would be realized according to the value of an attribute (attached to the entities) if *Based on Attribute* → *Type* field). The entities at the origin of the synchronization are then routed to their corresponding outputs (*No Batch* → *Batch Action after Matching* field).



Name: Match 1
Number to Match: 2
Type: Any Entities
Batch Action after Matching: No Batch
Comment:

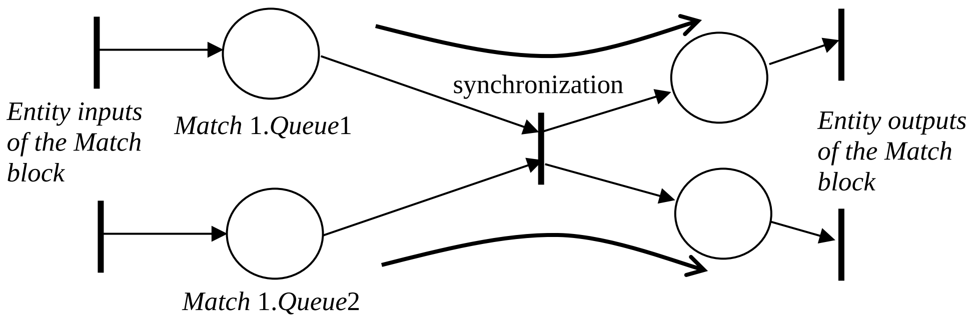


The ***Queue*** block (from template *Data Definition*, belonging to the *experiment part*) described below defines the queues *Match* 1.*Queue*1 and *Match* 1.*Queue*2.



	Name	Type	Shared	Report Statistics
1	Match 1.Queue1	First In First Out	☐	☑
2	Match 1.Queue2	First In First Out	☐	☑

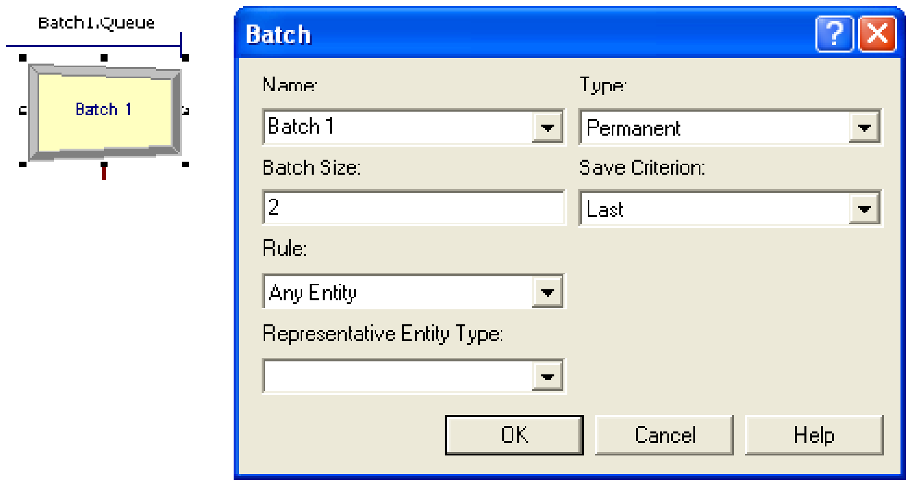
The following Petri net describes the behavior of the ***Match*** block *Match* 1.


synchronization
Entity inputs of the Match block
Match 1.Queue1
Match 1.Queue2
Entity outputs of the Match block

Note that all outputs of a ***Match*** block must be connected to a block (possibly a ***Dispose*** block if the output is not used).

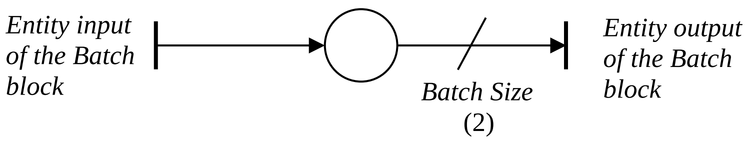
i) ***Batch*** (from template *Grouping*, belonging to the *model part*): A ***Batch*** block allows entities to be grouped together into a *representative entity*. A *representative entity* is an entity that consists of a group of entities that can travel together and be processed as if there was only one entity (e.g., a pallet of products); it can have a new *Entity Type* (see *Representative Entity Type* field).
The *representative entity* can be *temporary* or *permanent* (it depends on the value (*Temporary* or *Permanent*) of the *Type* field). In the case of a *temporary representative entity*, the ***Separate*** block provides the functionality to split/separate apart the representative entity into the individual entities which is not possible for a *permanent representative entity*.
The number of entities needed to form a group is specified in the *Batch Size* field. An entity arriving in a ***Batch*** block is held in a queue (associated with the block) as long as the number of entities accumulated in the queue is not sufficient to perform a grouping before they continue their movement with the *representative entity* in the model.
In addition, batches might be formed based on entities that have the same attribute when the *Rule* field is assigned to the *By Attribute* value, otherwise (that is, when batches are made without considering an attribute) the *Rule* field is assigned to the value *Any Entity* (which is the case by default).

For example, the *Batch* 1 block (*Batch* 1 $\rightarrow$ *Name* field) described in the following figure allows two entities to be grouped together into a *permanent representative entity*:



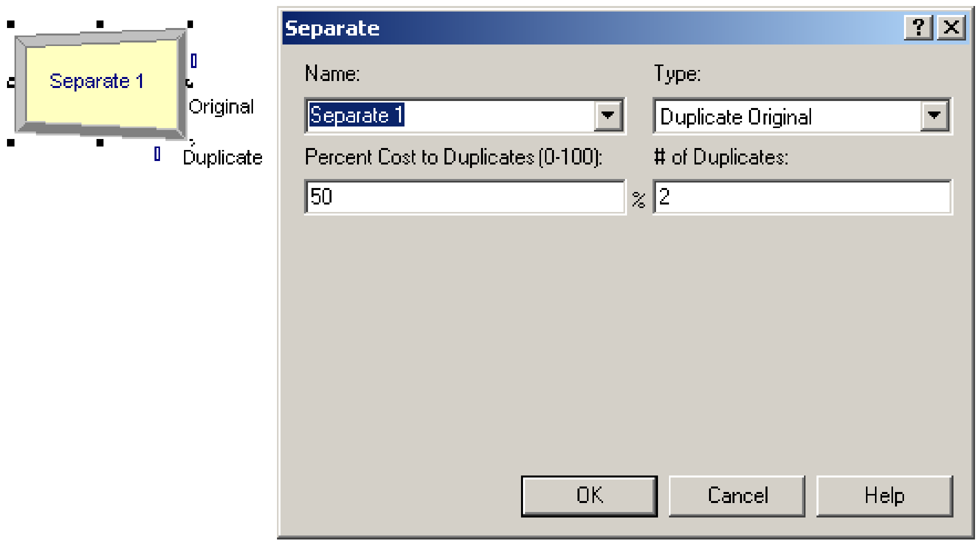
Name: Batch 1
Type: Permanent
Batch Size: 2
Save Criterion: Last
Rule: Any Entity
Representative Entity Type:

The following Petri net describes the behavior of the ***Batch*** block *Batch* 1.


Entity input of the Batch block
Batch Size (2)
Entity output of the Batch block

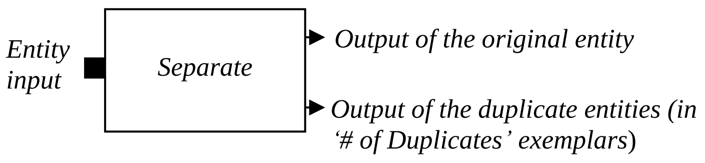
j) ***Separate*** (from template *Grouping*, belonging to the *model part*): A ***Separate*** block is used to clone/duplicate entities when *Duplicate Original* → *Type* field, the number of created entities is then specified in the '*# of Duplicates*' field. When an entity enters this block and has attributes, the attributes of all duplicate entities are the same as the current values of the attributes of the entity to be duplicated. The original entity exits through the *Original* output, the *# of Duplicates* entities (the duplicated ones) exit through the *Duplicate* output.
Note that a ***Separate*** block also allows the splitting of existing batch when *Split Existing Batch* → *Type* field.

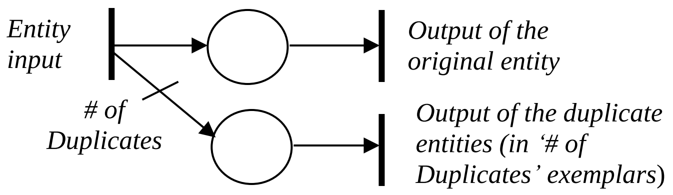
For example, a ***Separate*** block, entitled *Separate* 1 (*Separate* 1 → *Name* field), is described in the following figure. Do not worry about the text field in the dialog box relating to cost attributes.



Name: Separate 1
Type: Duplicate Original
Percent Cost to Duplicates (0-100): 50 %
# of Duplicates: 2

The following Petri net describes the behavior of the ***Separate*** block.


Entity input
Separate
Output of the original entity
Output of the duplicate entities (in '# of Duplicates' exemplars)


Entity input
# of Duplicates
Output of the original entity
Output of the duplicate entities (in '# of Duplicates' exemplars)

Another example is given in **\Examples\Separate\Separate.doe**.

k) *Process* (from template *Discrete Processing*, belonging to the *model part*): A ***Process*** block is used to simulate the behavior of basic processes such as a machine or a bank counter, knowing that different modes of functioning are possible depending on the content of the *Action* field (located in the *Logic* frame) when *Standard* → *Type* field.

When the *Action* field contains the value:

i) *Seize Delay Release*, the ***Process*** block corresponds to the blocks *Seize, Delay, Release* put in series (see VII.2.c,d,e). It allows the simulation of a process (machine, bank counter) requiring one, or more, resource(s) (see the *Resources* frame to assign the type, or the number, of resources concerned) during a *minimum* time (relative to the processing time) indicated in the *Delay* frame. Beyond this time the seized resource(s) are released (operation done in the *Release* block).
The equivalent Petri net corresponds to a concatenation of the ones described in VII.2.c,d,e.

2i) *Delay*, the ***Process*** block is simply a *Delay* block which allows the simulation of a processing time (see the *Delay Type* frame to assign a processing time), for example, of a machine. Note that with such a simplification the number of resources of the process is assumed to be infinite.
The equivalent Petri net corresponds to the one described in VII.2.c.

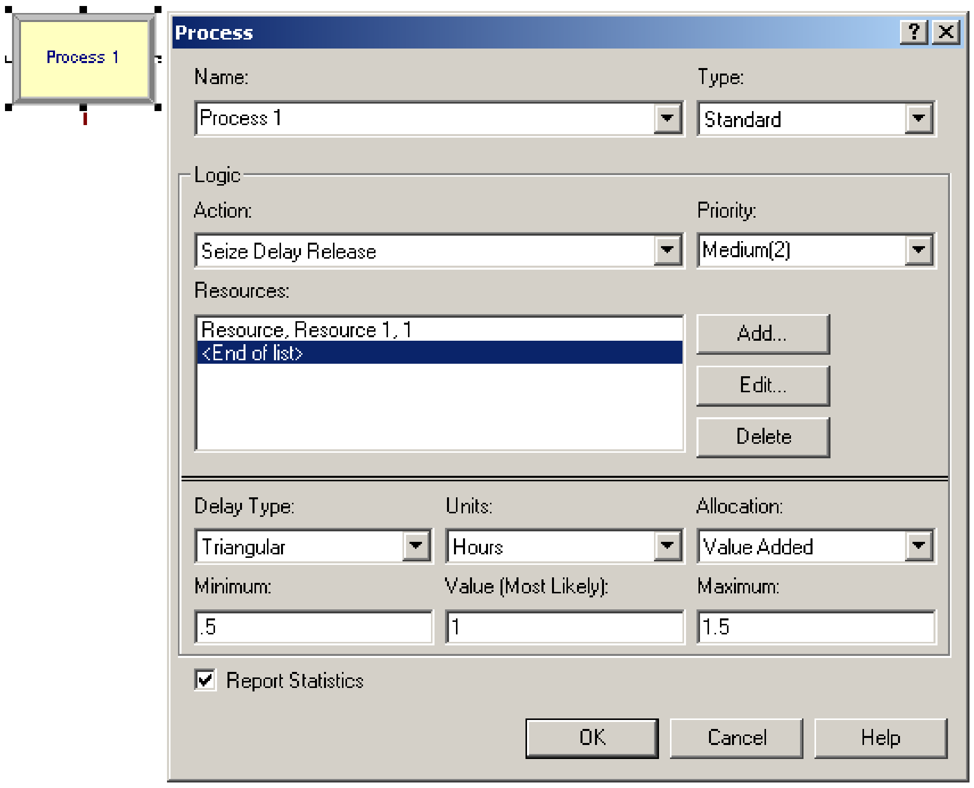
3*i*) *Seize Delay*, the ***Process*** block is reduced to the blocks *Seize* and *Delay* put in series. It allows the simulation of a process requiring one, or more, resource(s) (see the *Resources* frame to assign the type, or the number, of resources concerned) during a *minimum* time (relative to the processing time) indicated in the *Delay* frame. Note that the behavior of this block corresponds to the one of the previous case *i*) without the release of the resource(s) assumed to be realized downstream.
The equivalent Petri net corresponds to a concatenation of the ones described in VII.2.c,d.

4*i*) *Delay Release*, the ***Process*** block is reduced to the blocks *Delay* and *Release* put in series. It allows the simulation of a process requiring one, or more, resource(s) (see the *Resources* frame to assign the type, or the number, of resources concerned) during a *minimum* time (relative to the processing time) indicated in the *Delay* frame. Note that the behavior of this

block corresponds to the one of the previous case i) without the seize of the resource(s) assumed to be realized upstream.
The equivalent Petri net corresponds to a concatenation of the ones described in VII.2.c,e.

N.B.: The use of actions *Seize* or *Release* needs to also manage the *Queue* and *Resource* blocks (these blocks belonging to the *experiment part*).

For example, a ***Process*** block, entitled *Process* 1 (*Process* 1 → *Name* field) with *Seize Delay Release* → *Action* field, is described in the following figure.



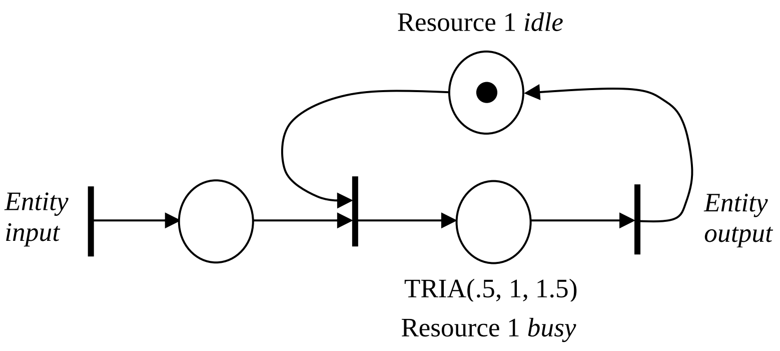
Name: Process 1
Type: Standard

Logic
Action: Seize Delay Release
Priority: Medium(2)
Resources:
Resource, Resource 1, 1
<End of list>

Delay Type: Triangular
Units: Hours
Allocation: Value Added
Minimum: .5
Value (Most Likely): 1
Maximum: 1.5

[✓] Report Statistics

The following Petri net describes the behavior of the *Process* 1 block.



VII.3 Blocks to Analyze Model

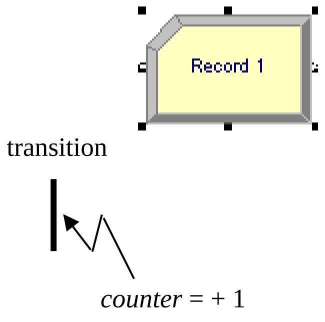
The blocks described in VII.2 allow the modeling of real-life systems by providing information given by default in the final report at the end of a simulation. The collection of extra information is done using additional blocks. Some of these blocks are described below, namely ***Record***, ***Counter***, ***Tally*** and ***Time Persistent***. An example is given in the **\Examples\Record_Statistic\Stat_File.doe** file.

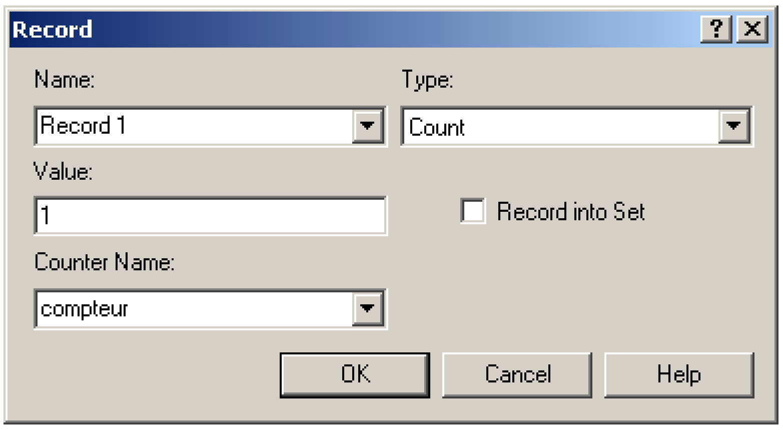
The ***Record***, ***Counter*** and ***Tally*** blocks, described in a first part, are used to extract information located at particular points of the block diagram. The ***Time Persistent*** block, described in a second part, is used to collect statistics provided by ARENA variables.

1) Block ***Record*** (from template *Input Output*, belonging to the *model part*) allow, according to the content of the *Type* field, to collect statistics (as average, minimum, maximum) on:

- the number of entities crossing the ***Record*** block when *Count* → *Type* field, the use of a ***Counter*** block (from template *Input Output*, belonging to the *experiment part*) allows the recording along the simulation time of these data in a file, see **1.a**;
- the time lapse between the successive passage of 2 entities in the ***Record*** block when *Time Between* → *Type* field, the use of a ***Tally*** block (from template *Input Output*, belonging to the *experiment part*) allows the recording along the simulation time of these data in a file, see **1.b**;
- the times taken by the entities to cross a part (or the whole) of the simulation model when *Time Interval* → *Type* field, the use of a ***Tally*** block allows the recording along the simulation time of these data in a file, see **1.c**.

1.a) When *Count* → *Type* field, the ***Record*** block allows the counting of the number of entities passing through the block: the name of the *counter* is specified in the *Counter Name* field*;* the *counter* increments by a value (1 by default) specified in the *Value* field each time an entity passes through the block.




Record

Name: Record 1

Type: Count

Value: 1

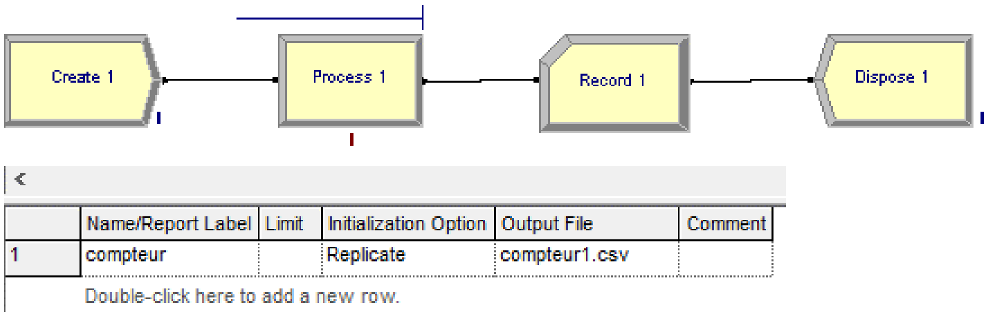
☐ Record into Set

Counter Name: compteur

The block ***Counter*** (from template *Input Output*, belonging to the *experiment part*) allows the recording during the simulation of the occurrence numbers *and* the passing times of entities going in each counter defined in a *Record* block used in the simulation model. To do this, complete the ***Counter*** block by indicating the name of the counter concerned *and* the name of the file (with its directory) in which the data will be saved.

N.B.: To save the data in a *csv* file (*comma-separated-value* ‘valeur séparée par une *virgule*’), recognized in particular by *MatLab* (through the instruction *CSVREAD*) and *Excel*, you should: put the extension *.csv* to the file *and* indicate that the file is in text format by checking the *Write Statistics Output Files as Text* box (accessible in the *Run/Setup.../RunControl/ Advanced* menu).

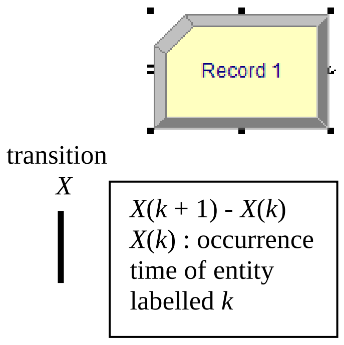
For example, in the following figure, the *compteur* 1*.csv* file contains the data collected by the *compteur* counter defined in the *Record* 1 block.

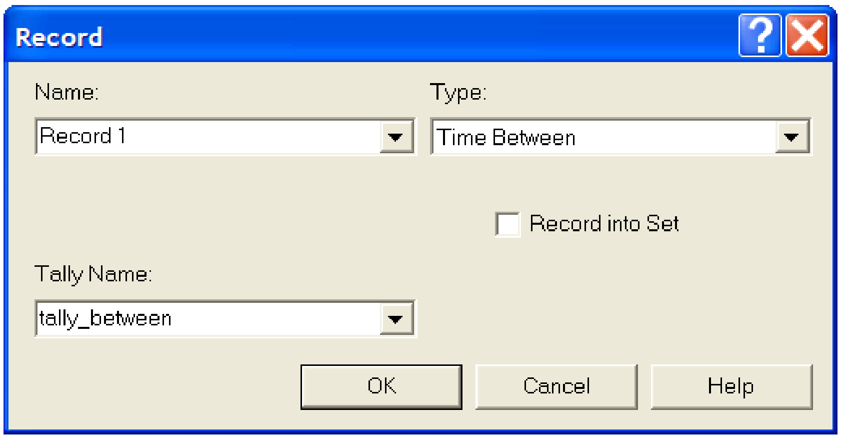


	Name/Report Label	Limit	Initialization Option	Output File	Comment
1	compteur		Replicate	compteur1.csv	

Double-click here to add a new row.

1.b) When *Time Between* → *Type* field, the ***Record*** block allows to collect the time lapse between the successive passage of 2 entities in the block. The name of the *tally*[5] (in charge of identifying this data) is specified in the *Tally Name* field.





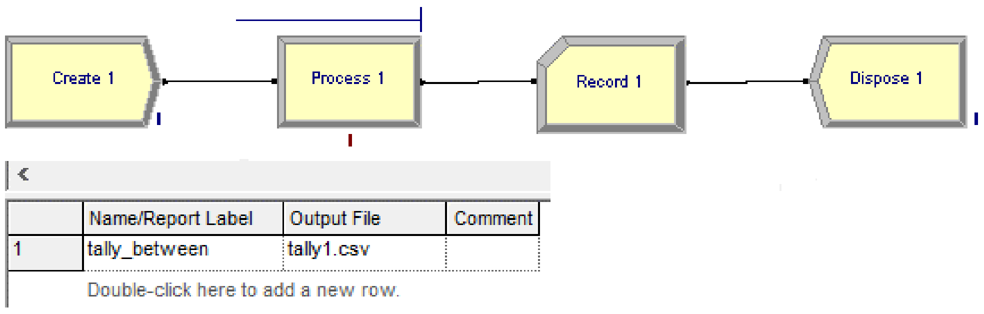
Name: Record 1
Type: Time Between
Record into Set: ☐
Tally Name: tally_between

The block ***Tally*** (from template *Input Output*, belonging to the *experiment part*) allows the recording during the simulation of the values of a *tally* (defined in a *Record* block). To do this, complete the ***Tally*** block by indicating the name of the *tally* concerned *and* the name of the file (with its directory) in which the data will be saved.

N.B.: To save the data in a *csv* file (*comma-separated-value* 'valeur séparée par une *virgule*'), recognized in particular by *MatLab* (through the instruction *CSVREAD*) and *Excel*, you should: put the extension *.csv* to the file *and* indicate that the file is in text format by checking the *Write Statistics Output Files as Text* box (accessible in the *Run/Setup.../RunControl/ Advanced* menu).

For example, in the following figure, the *tally*1*.csv* file contains the data collected in the *tally_between* tally defined in the *Record* 1 block.

[5] '*To keep a tally of ...*' means '*tenir le compte de* ...'.

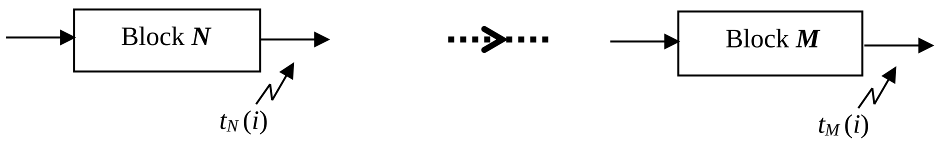


	Name/Report Label	Output File	Comment
1	tally_between	tally1.csv	

Double-click here to add a new row.

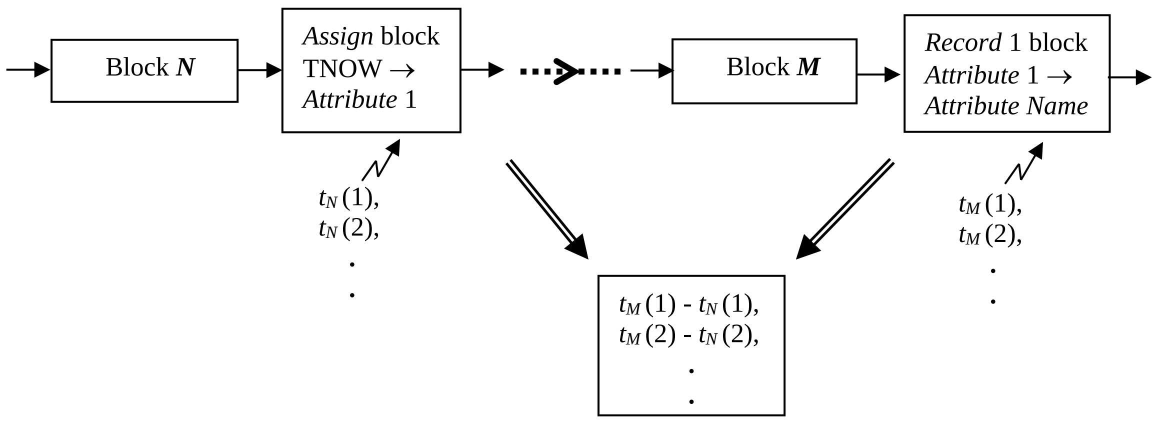
1.c) When *Time Interval* → *Type* field, the ***Record*** block allows to collect the times taken by the entities to cross a part (to define), or the whole, of the model. The name of the *tally* (in charge of identifying this data) is specified in the *Tally Name* field.

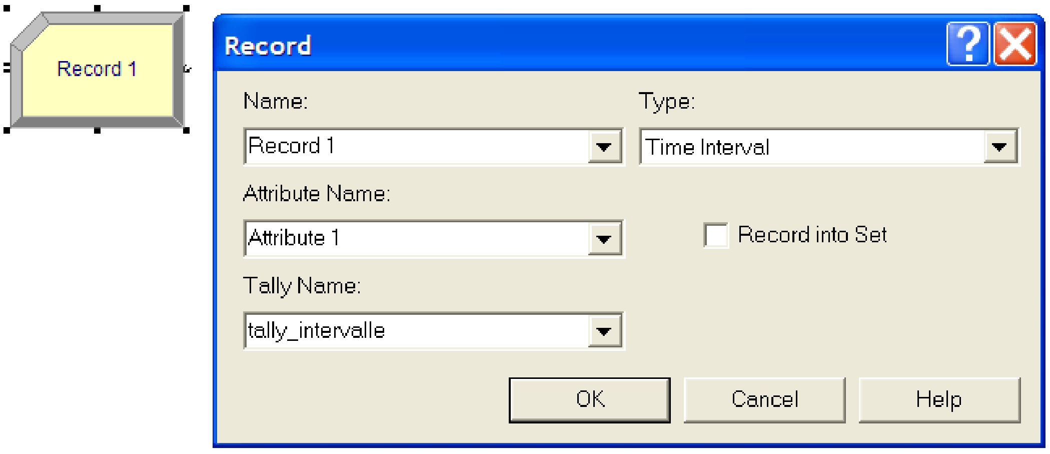
For example, suppose that we want to collect for each entity the difference between its output time from block ***M*** and its output time from block ***N*** (located upstream). Let $t_N(i), t_M(i)$ be the output times of entity labelled *i* of blocks ***N*** and ***M*** respectively, see the following diagram.



To do this, we include in the model:
- An ***Assign*** block (*cf. VII.2.f*) situated just after the output of the ***N*** block in order to define an *attribute*, noted for example *Attribute* 1, with the value *TNOW* to have the output time of the ***N*** block for each entity,
- A *Record* 1 block with *Time Interval* → *Type* field situated just after the output of the ***M*** block in order to have the travel time between the output of the ***N*** block and the output of the ***M*** block, the relation between this block and the *Attribute* 1 attribute (defined in the previous ***Assign*** block) is realized through the *Attribute Name* field.

See the following diagram to have a schematic view of the blocks ***N***, ***Assign***, ***M***, ***Record*** and the following figure where is described the *Record* 1 block.



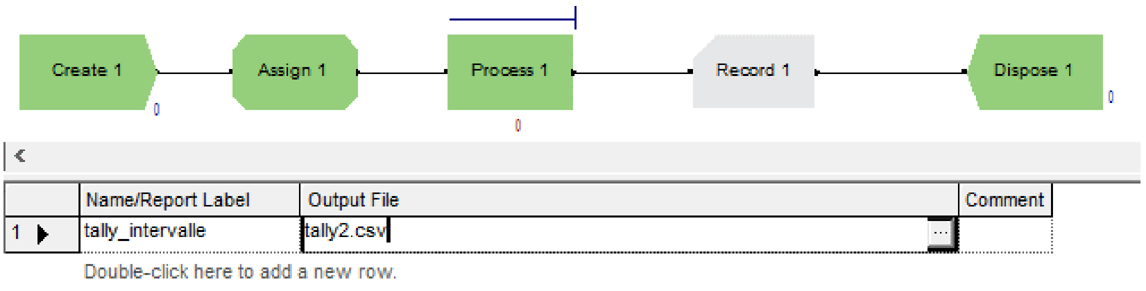


Name: Record 1
Type: Time Interval
Attribute Name: Attribute 1
Record into Set: ☐
Tally Name: tally_intervalle

The block ***Tally*** (from template *Input Output*, belonging to the *experiment part*) allows the recording during the simulation of the values of a *tally* (defined in a *Record* block). To do this, complete the ***Tally*** block by indicating the name of the *tally* concerned *and* the name of the file (with its directory) in which the data will be saved.

N.B.: To save the data in a *csv* file (*comma-separated-value* 'valeur séparée par une *virgule*'), recognized in particular by *MatLab* (through the instruction *CSVREAD*) and *Excel*, you should: put the extension *.csv* to the file *and* indicate that the file is in text format by checking the *Write Statistics Output Files as Text* box (accessible in the *Run/Setup.../RunControl/ Advanced* menu).

For example, in the following figure, the *tally*2.*csv* file contains the data collected in the *tally_intervalle* tally defined in the *Record* 1 block.



	Name/Report Label	Output File	Comment
1	tally_intervalle	tally2.csv	

2) The block ***Time Persistent*** (from template *Input Output*, belonging to the *experiment part*) allows to collect statistics provided by ARENA variables (automatically updated) such as the number of entities in a queue or the occupancy rate of a resource.

The variable ***NQ*** (*NQ* for *Number in Queue*) allows to have the number of entities in a queue. For example, the variable *NQ*(*Process* 1.*Queue*) allows to have the number of entities in the queue *Process* 1.*Queue*.

The variable ***NR*** (*NR* for *Number of busy Resource units*) allows to have the occupancy rate of a resource. Consider a machine with a processing capacity of *n* (that is, it can process *n* workpieces simultaneously) knowing that a resource can be *busy*. For example, consider a machine modeled by a ***Process*** block named *Process* 1, the machine has a resource named *Resource* 1 with a processing capacity of 3 (data defined in a ***Resource*** block), then the *NR*(*Resource* 1) variable allows to have the occupancy rate of the resource, that is, the number 0, 1, 2 or 3 of *busy* resources *Resource* 1.

For example, by setting in the *Time Persistent* block:

2.a) A statistic, named *Etat_file_attente*, we can collect in the *SIMAN Summary Report* file (available at the end of the simulation) the average, the minimum, the maximum number of entities in the queue *Process* 1.*Queue* by setting:

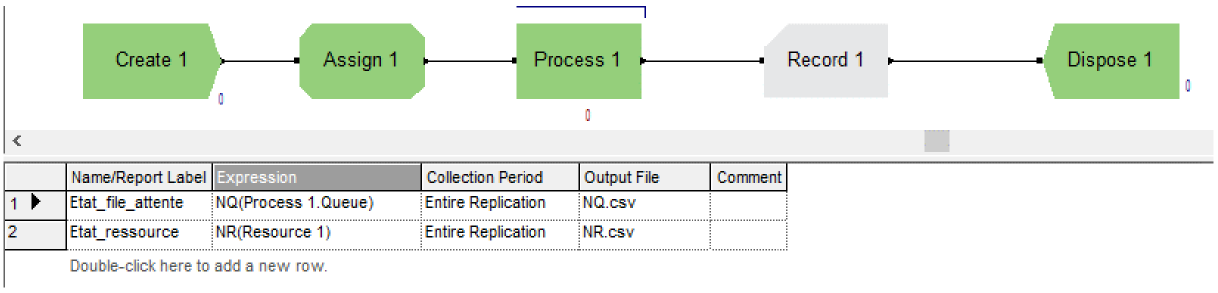
Etat_file_attente $\rightarrow$ *Name/Report Label* field,
NQ(*Process* 1.*Queue*) $\rightarrow$ *Expression* field, see the following figure.

2.b) A statistic, named *Etat_ressource*, we can collect in the *SIMAN Summary Report* file the average, the minimum, the maximum number of busy resources *Resource* 1 by setting:

Etat_ressource $\rightarrow$ *Name/Report Label* field,
NR(*Resource* 1) $\rightarrow$ *Expression* field, see the following figure.

In the same way, we can save this observation data, recorded during the simulation, by setting in the *Output File* field the name of the file (with its directory) in which data will be recorded. In the previous example, the statistics data *Etat_file_attente* and *Etat_ressource* are saved in the files *NQ.csv* and *NR.csv* respectively.

N.B.: To save the data in a *csv* file (*comma-separated-value* 'valeur séparée par une *virgule*'), recognized in particular by *MatLab* (through the instruction *CSVREAD*) and *Excel*, you should: put the extension *.csv* to the file *and* indicate that the file is in text format by checking the *Write Statistics Output Files as Text* box (accessible in the *Run/Setup.../RunControl/ Advanced* menu).



	Name/Report Label	Expression	Collection Period	Output File	Comment
1	Etat_file_attente	NQ(Process 1.Queue)	Entire Replication	NQ.csv	
2	Etat_ressource	NR(Resource 1)	Entire Replication	NR.csv	

Double-click here to add a new row.

VII.4 Graphic Animation

Animation is a effective way of communication, it brings a simulation model to life by generating a moving picture of model functioning, we can see the execution of the model, the entities as they wait in queues, occupy resources, travel between blocks, and so on. Easy to use, especially for decision-makers who are not necessarily initiated to technical aspects, the animation allows (provided that the simulation conditions are credible) to highlight the studied phenomena.
Use of animation during the building of a simulation model is also interesting to verify its functioning through a step-by-step visualization (see command *Step* ▶|) of the entities flow in the simulation model.

The objects available in ARENA allow the construction of a detailed graphical representation of the simulated system. This representation changes during simulation execution to reflect the movement of entities through the model (for example, the workpieces circulating through a production line) and the corresponding changes in the state of the system, in particular through a visualization of:

- the status of resources, for example, the *idle* or *busy* state of a machine or a robot,

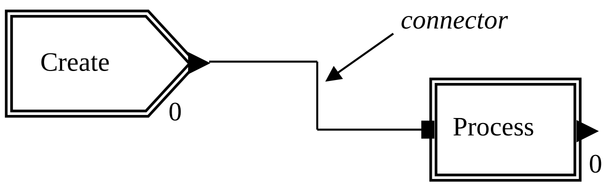
- the status of queues, for example, the buffer of a machine,
- the evolution of the values of variables or attributes, for example, a variable indicating the processing time of a machine.

There are two types of objects:
- *static objects* as lines, boxes, circles, they don't change during simulation execution and are typically used to represent the physical structure or background environment of the system,
- *dynamic objects* change shape, color, or position during the simulation execution, for example, workpieces, workers, machines are represented as dynamic objects. Such objects also include the graphical representation of the values of system variables and summary statistics.

An example is given in the file **\Examples\Animation_Un_exemple\Animation.Doe**.

1. Animation of entities

An animation allows the visualization of the flow of entities along the connectors linking the modules between them:

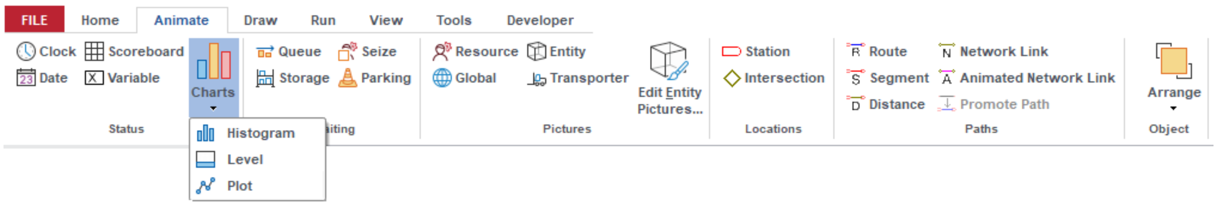


Note that the movement of the entities along the connectors has no impact on the simulation time (*TNOW*). The simulation of transport times requires the use of the block ***STATION*** of *Discrete Processing* template.

The *initial image* used to represent an entity is defined in the ***Entity*** block (from template *Data Definition*, belonging to the *experiment part*). The image is by default the one noted *Picture.Report* and is indicated in the *Initial Picture* field of the ***Entity*** block. The change of the image associated to an entity is done *via* its passage in an ***Assign*** block by assigning a new image to the *Entity Picture* attribute (for example, *Entity Picture* $\rightarrow$ *Type* field, *Picture.Truck* $\rightarrow$ *Entity Picture* field).
The *Animate/Edit Entity Pictures* menu provides access to other images (included in *.plb* files). The *Open* button opens a particular *.plb* file, for example, the *Equipment.plb* file provides images of various machines.
It is also possible to create your own images (saved in a *.plb* file). Click on the *Add* button on the right to create an image, which makes a gray box appear. Double-clicking in this box opens a window (*Picture Editor*) where it is possible to design an image. Close the window once the image is designed; it will then appear instead of the grey box. The *Save* button allows the recording of created image(s) in a *.plb* file of your choice.

➢ The Animate toolbar

The **Animate** toolbar provides an interface with the basic objects of the animation.



Some of these objects are described below:

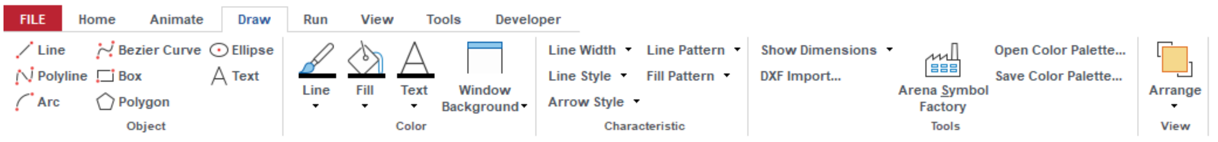
- *Status* views:
  - *Clock* displays the simulation time in hours, minutes, and seconds,
  - *Date* displays the simulation time (*TNOW*) in days, months, and years,
  - *Variable* displays the numerical value of a mathematical or logical expression,
  - *Histogram* displays the distribution of the value of an expression in a specified range,
  - *Level* displays the value of an expression relative to specified minimum and maximum values,
  - *Plot* displays the past values of an expression over a specified time range.

- *Waiting* area:
  - *Queue* displays entities waiting for a specified event, for example, the availability of a resource.

- *Pictures*:
  - *Resource* allows to have an object (for example, a machine) with limited capacity that can be allocated to entities. A resource can be in an *idle* or *busy* state, the image associated with a resource during the simulation depends on its state,
  - *Global* allows the association of images to an expression (variable, attribute). The image associated with the expression during the simulation changes according to its value relative to a specified value.

➢ The *Draw* toolbar

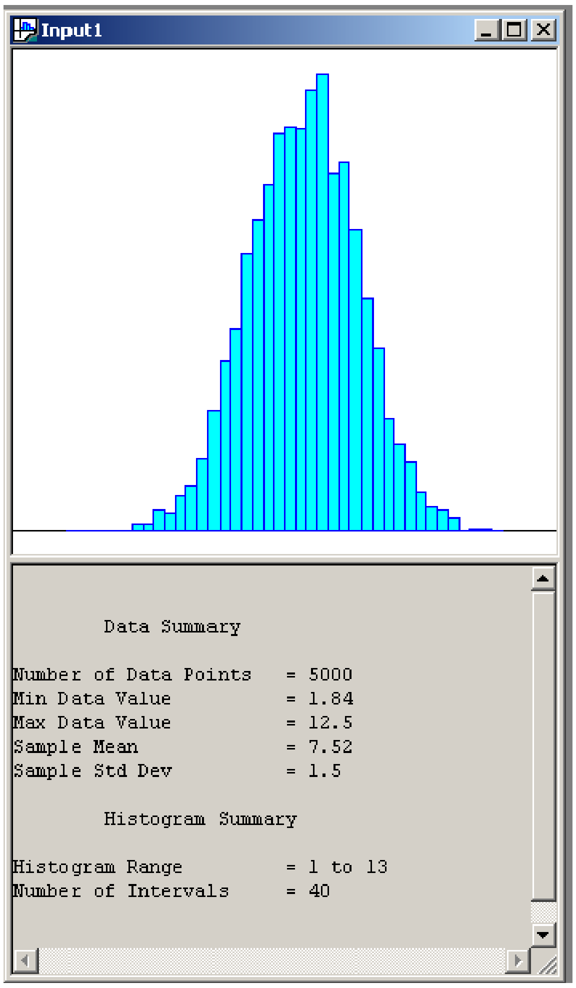
The objects of the **Draw** toolbar allow the addition of static drawings or text:



VII.5 INPUT ANALYZER

The *Input Analyzer* included with ARENA allows the exploitation of input data by determining the most representative probability distribution of the empirical distribution obtained from the input data.
The input data is contained in a *.dst* file in which each row contains a number. In fact, such file is a text file (readable, for example, by using *WordPad*). See, for example, the *test*1.*dst*, or *test*2.*dst* files accessible in the ***\Examples\Distribution*** directory. To analyze such a data file, select the *Tools/Input Analyzer* menu to open the *Input Analyzer* application. Once the application is opened, select *File/New* to open a window named *Input*1. After clicking in the

gray window, select *File/Data File/Use Existing…*[6] to select the *.dst* file containing the data to be converted into a distribution law, which leads to the display of a histogram of the data.



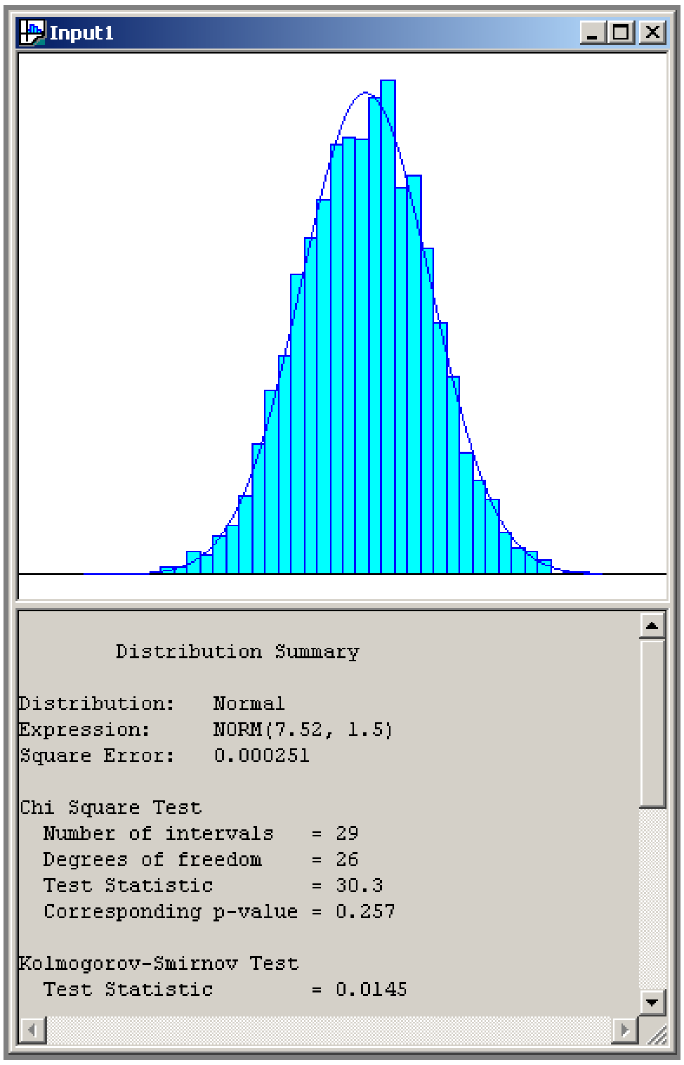
```
        Data Summary

Number of Data Points    = 5000
Min Data Value           = 1.84
Max Data Value           = 12.5
Sample Mean              = 7.52
Sample Std Dev           = 1.5

        Histogram Summary

Histogram Range          = 1 to 13
Number of Intervals      = 40
```

Information about the data (number of data, minimum and maximum values, mean, standard deviation) and the histogram (range ('portée') (smallest value and largest value considered in the histogram), the number of intervals (in $O(\sqrt{nbre\ de\ pts})$, between 5 and 40 intervals)) are described in the gray window.

Select *Options/Parameters/Histogram...* to change the parameters of the histogram, namely the *number of intervals*, the *lower bound* (data with lower values are ignored), and the *upper bound* (data with upper values are ignored).

Select *Fit/Fit All* ('Ajuster') to find the best distribution corresponding to this histogram.

[6] It is also possible to create a data file (*.dst*) corresponding to a certain distribution by selecting *Generate New....*



```
        Distribution Summary

Distribution:   Normal
Expression:     NORM(7.52, 1.5)
Square Error:   0.000251

Chi Square Test
  Number of intervals   = 29
  Degrees of freedom    = 26
  Test Statistic        = 30.3
  Corresponding p-value = 0.257

Kolmogorov-Smirnov Test
  Test Statistic        = 0.0145
```

VII.6 OUTPUT ANALYZER

Once the simulation execution finished, the *Output Analyzer* included with ARENA allows the calculation of statistical results as mean, standard deviation, *minimum* value, *maximum* value. It is also possible to collect the complete history of the values obtained during the simulation execution in files (usable for example by using the *Excel* software). In addition to being able to draw curves and histograms, the *Output Analyzer* offers different statistical functions as calculation of confidence intervals, construction of correlograms, comparison and analysis of means, analysis of variances, hypothesis testing.