



POLITECNICO DI TORINO

Master's degree in Mechatronic Engineering

DEVELOPMENT AND IMPLEMENTATION OF AN APPLICATION SOFTWARE FOR THE SIMULATION OF PACKAGING LINES

Supervisors:

Prof. Marina INDRI

Prof. Michele TARAGNA

Candidates:

Francesco Pranzo

Matteo Marangoni

Supervisors at SACMI:

Ing. Stefano Cocito

Ing. Franco Lumini

October 20

Index

INTRODUCTION	4
CHAPTER 1 -THE IMPORTANCE OF SOFTWARE	5
1.1 Simulation models pros and features	5
1.2 Insight of a simulation model	7
1.2.1 Main modelling elements	7
1.2.2 Model validation	8
1.2.3 Theoretical model verification	9
1.3 Focus on Plant simulation	9
1.3.1 Hierarchy of the objects	10
1.3.2 Output of the simulation	11
CHAPTER 2 -PLANTS GLOBAL OVERVIEW	13
2.1 Ridge distribution	14
2.1.1 Source	15
2.1.2 Pullnose	16
2.1.3 Control logic	16
2.1.4 Pivot	17
2.1.5 Reject zone	17
2.1.6 Chart	18
2.1.7 Chart with double pivot for recycling	20
2.1.8 Monitoring and management of rank positioning	20
2.1.9 Pitch operating mode	21
2.2 Legs and flowpacks	22
2.2.1 TT zone	23
2.2.2 Orienting zone (EPO)	24
2.2.3 Accumulating zone	26
2.2.4 Speed variations	27
2.2.5 Optimal length of the belts in accumulating zone	27
2.2.6 Speed regulation in the flowpack	28
2.2.7 Classic speed regulation	29
2.2.8 Speed regulation with compensation	29
2.2.9 Passive gap closing	30

2.3 Buffer and recycling	30
2.3.1 Double recycling	31
CHAPTER 3 -THE CONCEPT OF CLASS	35
3.1 Class “PIVOTSX”	38
3.1.1 “D_P_X” Methods	40
3.1.2 “NI_X” Methods and Attributes	41
3.2 Class “PIVOTDX”	42
3.2.1 “D_P_X” Methods	44
3.2.2 “D_R_X” Methods	44
3.3 Class “LEGX”	46
3.3.1 “OR_1” and “Or_2” Methods	49
3.4 Class “FLOWPACKX”	49
3.5 Class “PULLNOSEX”	51
3.5.1 “P_P1” Methods	53
3.6 Class “INTERASSEBLX”	53
3.7 Class “BUFFERGX”	55
3.7.1 “D_BUFFER” Methods	60
3.7.2 “BUFFER” Methods	61
3.7.3 “ESTRA_SUP” and “ESTRA_INF” Methods	62
3.8 Class “BUFFERVX”	62
3.8.1 “BUFFER” Methods	66
3.8.2 “USCITABUFFER_X” Methods	66
CHAPTER 4 -INPUT AND OUTPUT MANAGEMENT	67
4.1 Input management	68
4.1.1 Frame controls content	68
4.1.2 Insight in the objects classes: reset method	77
4.1.3 Insight in the objects classes: init method	80
4.2 Output management	83
4.2.1 Controls frame: output portion	84
4.3 Failures management	88

4.3.1 Insight in the flowpackx	88
4.3.2 Insight in the buffers	92
CHAPTER 5 -TESTS AND ANALYTICAL RESULTS	94
5.1 Test on the first distribution	94
5.2 Test on the second distribution	104
CHAPTER 6 -CONCLUSIONS AND FUTURE DEVELOPMENTS	116
BIBLIOGRAPHY	118

Introduction

This dissertation is about the implementation of some of the plants developed by Sacmi Packaging & chocolate in a software powered by Siemens, called Tecnomatix PlantSimulation.

In depth, this software allows to simulate the plants, by implementing their characteristics and functionalities. Performing simulations is a crucial aspect that allows the factory to save money not necessarily testing some features on the field, but only by simulating them in a virtual environment, given by this kind of software.

The software is provided with some predefined objects that are then suitably programmed and modified in order to make them work out their functions according to the ones that shall be provided by the considered plant.

Some models of the principal components of these plants are built and stored in a library, in order to allow the user just to drag and drop these different models into a blank frame, building in this way the plant that must be considered.

In the first chapter of this thesis the importance of the software is highlighted according to the possibility of performing any simulation on the considered plant and its main characteristics are explored.

The second chapter concerns the configuration of the real plants, so how they work and how they are structured.

The third chapter is dedicated to the objects built in the library: by suitably programming and modifying the standard objects already present in the software, it is possible to build many different plants of interest and then create a personal library with all the objects in it representing specific elements of the different lines considered.

The fourth chapter concerns the different management of input and output data, in order to suitably set the needed data to perform the simulation and at the end of it collect all the data of interest.

Finally, in the fifth chapter, an analysis of the obtained results is performed and possible future developments are suggested in the sixth and last chapter.

Chapter 1 - The importance of software

A system is a set of elements which depend on each other. This dependence can be described through some variables linked one to the other by some mathematical laws that allow to compute the value of a variable when other variables or parameters are varying.

When a system complexity grows, it is not easy to find the laws that allow to derive the values of the critical variables. In particular, a system is said to be complex when it is made of so many elements that make difficult the comprehension of the system itself.

In these situations, it could be helpful using an empirical approach supported by suitable simulation and modelling tools.

A model can be defined as a tool needed to rebuild a certain system. In depth, different kinds of models exist:

- Mathematical models;
- Physical models;
- Simulation models.

Mathematical models are made of analytical laws that regulate the considered system.

Physical models are reconstructions of some systems aimed at studying the particular behaviour of the system rebuilt.

Simulation models are a simplified representation of a system able to analyse the system itself in all its states.

In this thesis, the focus is on simulation models.

1.1 Simulation models pros and features

Simulation models allow to study a certain system from different perspectives and reach more or less accurate results, according to the level of accuracy with which the model has been built.

The principal advantages of making use of simulation models are the following:

- Performing many experiments on the real system is not always possible, since it could be damaged;
- The cost of a simulation on the models is much cheaper than the cost of a real experiment led on the real system;
- The model helps to understand the system by putting in evidence only its essential aspects.

On the other hand, the cost to build a simulation model is not always so low and the times taken to build these kinds of models could be large. Moreover, a too simplified model could lead to errors while simulating the considered system [1].

In order to avoid these problems, many factories have implemented in the simulation tools some specific libraries appropriate for the specific industrial environment with which the factory deals.

Furthermore, other classifications of the models exist. In fact, there can be also considered:

- Static models;
- Dynamic models.

Static models are the ones that are not affected by the time variable. Dynamic models, instead, depend on the time and they are guided by events: when an event occurs, the different values of the variables change and also the state of the models changes.

Plant simulation powered by Siemens is a dynamic simulation software, since it analyses a system in a certain simulation time that corresponds to a much wider effective time.

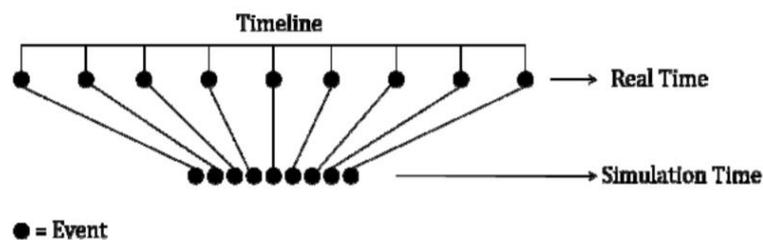


Figure 1.1 Simulation time vs effective time [2]

Other kinds of models to be considered are:

- Deterministic models;
- Stochastic models.

A stochastic model contains one or more variables coming from random distributions, which do not assume a single value, but a range of values.

A deterministic model, instead, is a model with no random variables and this kind of model does not allow to perform an accurate analysis of the perturbations acting on the system; the results, in fact, are referred to mean values, not taking into consideration their maximum and minimum values or their standard deviations.

Finally, there are also:

- Continuous time models;

- Discrete time models.

In the first ones, the variables change continuously in time. In the latter, the values of the variables are defined at each event processed by the system.

Discrete time simulation models are called DES (discrete event simulation) and Plant simulation belongs to this category.

1.2 Insight of a simulation model

Simulation is a technique that allows to reproduce the operations a system performs by building a suitable model of the system itself.

As already said, a model is a simplification of the reality, since obtaining a too accurate model would be complex and too expensive.

1.2.1 MAIN MODELLING ELEMENTS

In a generic simulation software, these elements are always present:

- **Controller:** manages the sequence of events and the evolution of the system state in time, by allowing the flow of the simulation time. In Plant simulation this role is played by the Event Controller.
- **System:** set of entities and resources that constitute the objects of the system.
- **Entities:** elements of the system that flow in the system itself (e.g. pieces, parts). In Plant simulation these are the mobile units (MU).
- **Resources:** components of the system that are allocated to entities, such as machines. The most used in Plant simulation are the single processes, so machines that perform a single activity on the different entities.
- **Attributes:** values associated to single entities or to resources, such as the working time, arrival time. In Plant simulation these could be already present in the object (standard) or defined by the user (user defined attribute).
- **Variables:** could be parameters or support variables used in order to make the communication among different objects easier.
- **Events:** time instant in which a change in the system state happens. Time changes according to the passage from an event to the subsequent one.
- **Activity:** every object in the system performs an activity, which is something that takes place in between two different events and corresponds to the status of one or more entities, requires a certain amount of time to be completed and determines a modification to the system state.

- **Processes:** are sequences or predefined cycles of activities. In DES the running process is not visible in its sub-activities, but only its states can be observed: in this way, a station will result “blocked”, “working” or “waiting”, but it is not possible to know the percentage of the work done. Plant simulation gives this kind of feature in the section “Statistics” of each object.
- **State:** specific configuration in which the system is [2].

1.2.2 MODEL VALIDATION

The validation of the model implies that the results given by the model correspond to the ones given by the real system. So, if the results of the model differ too much from the ones given by the real system, the model created results to be not valid and it must be suitably corrected.

Moreover, in addition to the model validation that is performed only once, also the verification of the model must be performed. This consists in determining if the simulation model works as expected by who has developed it. This verification is not unique and it is performed according to the analyst opinion.

Usually, these verifications are performed by making some proofs either by making use of known results or through the verification that all the most important elements have been included in the model; otherwise, simulations with deterministic values are run in order to obtain results easy to verify.

Plant simulation also allows to perform a dynamic validation of the process by observing it during its execution. In fact, the icons of the single processes change colour according to the different state of the object:

State of the object	Looks like this	LED	State icon
Failed		red	Failed
Stopped		pink	—
Paused		blue	Pause
Unplanned		light blue	—
Working		green	Working
Blocked		yellow	—
Setting-Up		brown	Setup
Powering up/down		purple	—

Table 1.1 Colours due to the different states [2]

1.2.3 THEORETICAL MODEL VERIFICATION

In order to verify the theoretical validity of the model, so the fact that in absence of failures it works at 100% of the efficiency, it is necessary to compute the efficiency in the following way:

$$Efficiency = \frac{Throughput\ per\ hour * 100}{\frac{3600}{TC}} \quad (1.1), \quad where$$

- TC = cycle time.

The hypothesis to be considered to test the model in these conditions are the following:

- No machine failures;
- Single process machines at TC (cycle time);
- Warmup time must be considered (time between the start of the simulation and the moment in which the statistics can start);
- The efficiency must result to be 100% [3].

1.3 Focus on Plant simulation

Tecnomatix Plant simulation is a discrete time simulation tool that allows to create models of production plants in a virtual environment, in order to analyse and optimize their performances [4].

The different production scenarios are analysed by considering the bottle necks of the plant, statistics and graphs. The results achieved are useful to take decisions since the preliminary phases of the planning.

In practice, Plant simulation gives a blank frame to the user in which the different elements that compose the production system can be arranged. Once these elements are dragged and dropped into the frame, they must then be connected one to the other in order to make them behave as expected in the real plant.

The dynamic execution of the simulation is managed by the Event controller, which is the clock of the software.

The operations that must be performed by the software are fed to it by means of the so called methods, which contain pages of executable code. The programming language used is SimTalk, similar to C language, but enriched with some elements such as the object oriented programming, which allows to interact with objects present in the modelling area [5].

The objects that can be inserted in the frame can be found either in the class library or in the toolbox, as reported in figure 1.2.

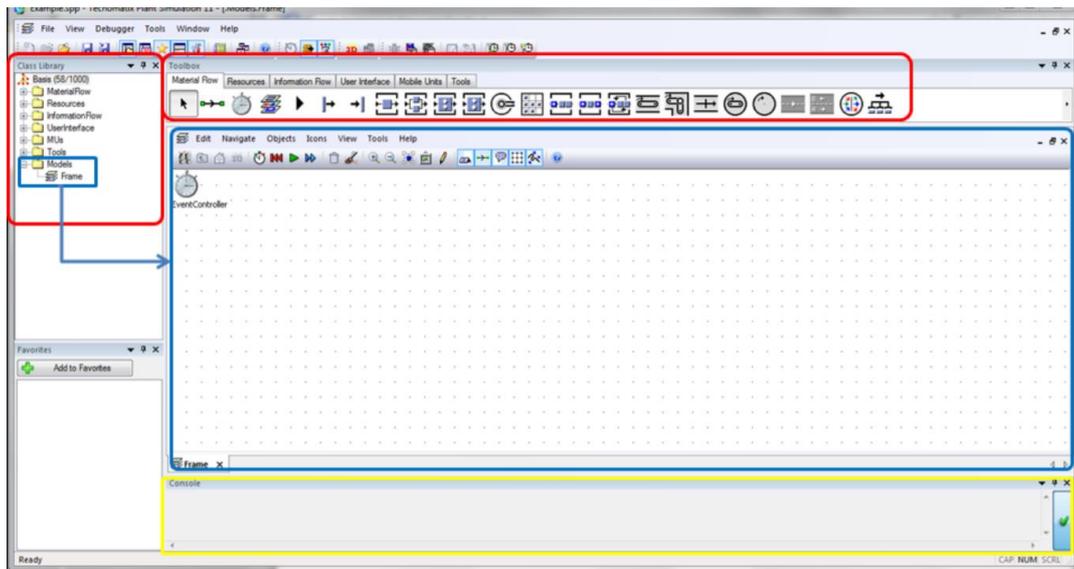


Figure 1.2 Blank frame [2]

Finally, the yellow area is the console in Plant simulation and here the actions performed during the execution of the simulation are printed [6].

Once the model is created, the simulation can be run by the play button and reset by the reset button:

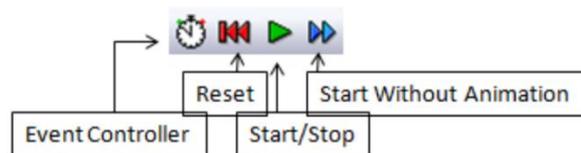


Figure 1.3 Main commands to start/stop the simulation [4]

Once the run has started, it is possible to observe a scaled representation of the real system in a virtual environment, provided with a really useful animation of the latter that allows to identify possible problems that could arise during the simulation.

1.3.1 HIERARCHY OF THE OBJECTS

When a large number of objects must be inserted in the frame, with many of them presenting similar features, building some classes of objects could result a really useful procedure.

The libraries, or toolbar, contain, in fact, the so called master classes, which are the standard objects for each type of object considered. Once this kind of object is inserted in the frame, it becomes an instance. In the same frame two instances with the same name

cannot coexist. The difference between a master class and an instance is that every modification apported to the master is automatically inherited by the instances, while if a modification is apported to the instances, this change only belongs to the specific instance and does not involve all the other instances of the same object. The inheritance can also be unselected, but the default setting is the one just mentioned.

Some of the objects present in this virtual environment are a representation of physical entities really existing in the real plant; other objects represent the logic connection used to move the mobile units, defining in this way the route on which the mobile units must travel. Finally, also graphs and tables are present in order to manage and get useful information when the simulation is performed [7].

1.3.2 OUTPUT OF THE SIMULATION

At the end of the simulation a report is automatically generated. In this report information related to the statistics of the model are contained [8].

The information achieved for each workstation is the following:

- Percentage of time while waiting for material;
- Percentage of time it has been busy;
- Percentage of time it has been blocked;
- Percentage of time a failure has occurred;
- Number of worked pieces;

Similarly, for each warehouse (e.g. buffers), the following information could be retained:

- Number of units entered and exited;
- Initial and final number of units laying in it;
- Maximum and minimum number of units laying in it during the simulation.

All this information is contained in the tab Statistics of each object present in the frame.

For each drain the software computes the following variables:

- Cycle time: mean time taken by each entity to go through the production line (called mean life time).
- Throughput per hour: number per unit of time of mobile units exiting the line, which are counted as final products.
- Percentage of time the entities remain in the different objects which constitute the production line [9].

Object	Name	Mean Life Time	Throughput	TPH	Production	Transport	Storage	Value added	Portion
No_Detrattori	Entity	14:00.0000	598	12	100.00%	0.00%	0.00%	100.00%	
Drain_Rilavorazioni	Entity	15:18.5217	460	9	100.00%	0.00%	0.00%	100.00%	
Drain_Setup	Entity	15:48.0000	450	9	100.00%	0.00%	0.00%	88.61%	

Figure 1.4 Cumulated statistics of the parts deleted by the drain [2]

Chapter 2 - Plants global overview

The plants considered in this dissertation are essentially two. They are quite similar for what concerns the types of machines by which they are composed, but they differ for the buffer structure and the structure of the lines located after it. Anyway, these two plants are essentially composed by the same blocks of conveyors, stations and buffers, that can be grouped, dividing the plants in the following segments:

- Ridge distribution;
- Legs and “Flowpacks”;
- Buffer and recycling.

In particular, the global overviews of the two plants are the following:

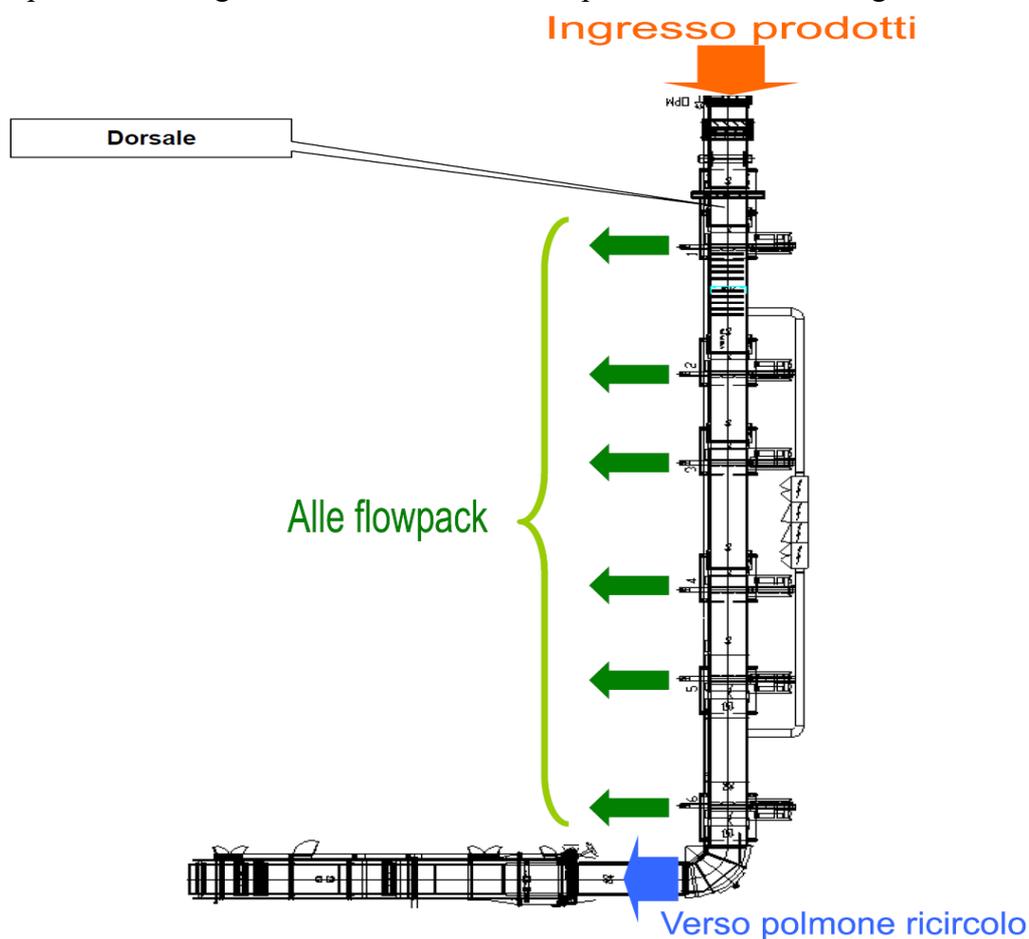


Figure 2.1 Plant with fan buffer [Credit: SACMI]

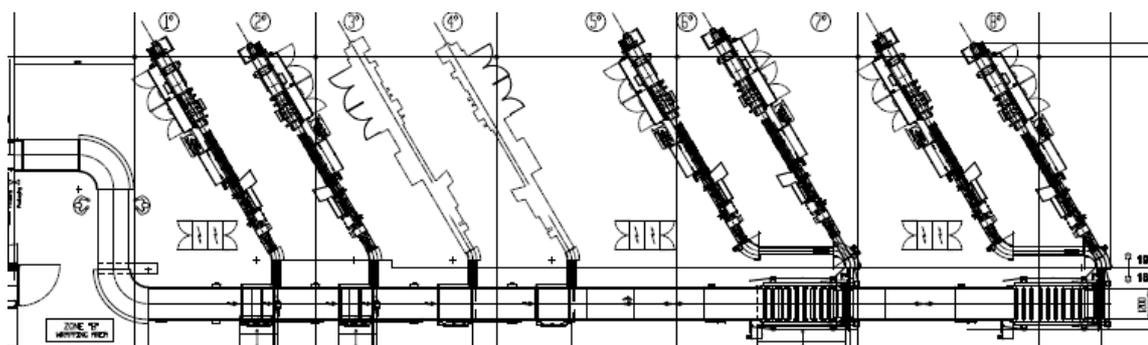


Figure 2.2 Plant with gondola buffer [Credit: SACMI]

2.1 Ridge distribution

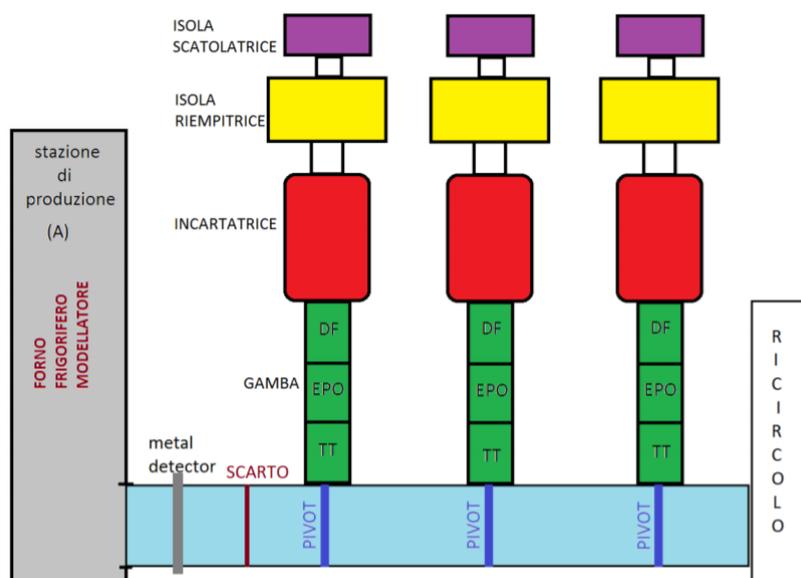


Figure 2.3 Ridge distribution global overview [10]

According to the behavior of the plant, the source A produces a certain quantity of product constant in time, placed in rows called ranks; these ranks are essentially rows of products, which are located one next to the other.

The ridge distribution (also called rank distribution) has the aim of distributing the ranks to the different legs, after that a quality check is performed; in this way some ranks are

discarded, while the ones that pass the quality check flow into the legs. Then, the legs provide these products to the Flowpack machines, that wrap the various products.

At the end of the ridge line, there could be a recycling system that has the functionality of keeping the products not given to the legs, in order to use them later in case of lack of products [10].

The details of the different components of the ridge distribution are analysed in the following subsection.

2.1.1 SOURCE

The source (which is generally an oven) cooks a large portion of product which is then cut. The finite product is then caught by the ridge distribution and, eventually, led to the legs.

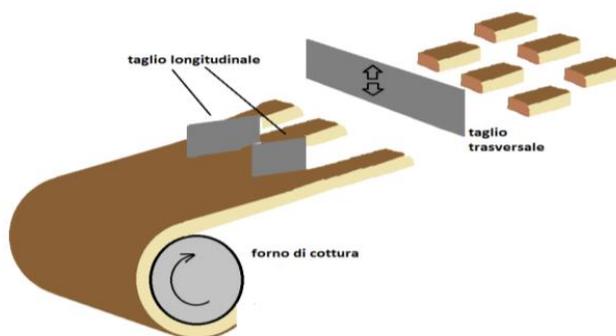


Figure 2.4 Typical source [10]

The source could be essentially of two different types:

- **Pitch movement:** moulds are treated with a regular timing, but in a discontinuous way. In this modality every k seconds the source provides two or more closed ranks separated by the next following two by a gap.
- **Continuous movement:** all the system moves continuously, with no speed variations. In such a way the ranks are already decoupled and equalized while exiting from the source.

In order to decouple and equalize the ranks coming from a source working with a pitch movement, a solution called Pullnose is adopted.

2.1.2 PULLNOSE

The Pullnose is a machine that is composed by two different belts plus a translator, which has an effect on the length of the belts connected to it. By suitably moving the translator, the gaps left by the source are filled.

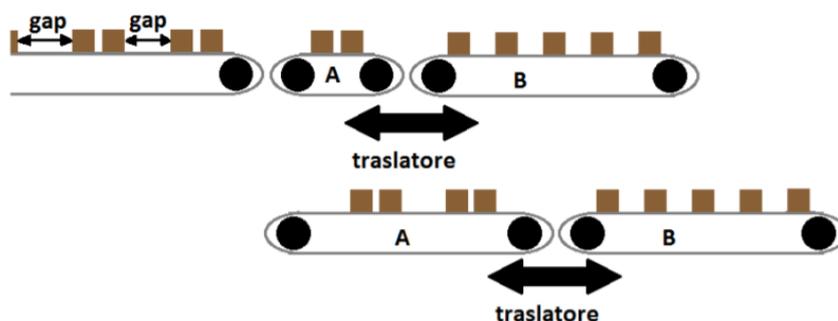


Figure 2.5 Pullnose structure [10]

2.1.3 CONTROL LOGIC

The speed is measured in hits/minute, where a hit corresponds to a rank. The speed of the belts, instead, is measured in meters/minute.

The typical speed of the ridge distribution is between 40 and 90 hits/min, while the standard belts velocity is 24 m/min.

The advancement of the distribution is controlled by a virtual phonic wheel, simulated by the PLC thanks to a suitable scaling of the instantaneous position value measured by an encoder of the motors of the belts. In the past the phonic wheels had a pitch configuration for which at every pulse the belt advanced of 6 mm.

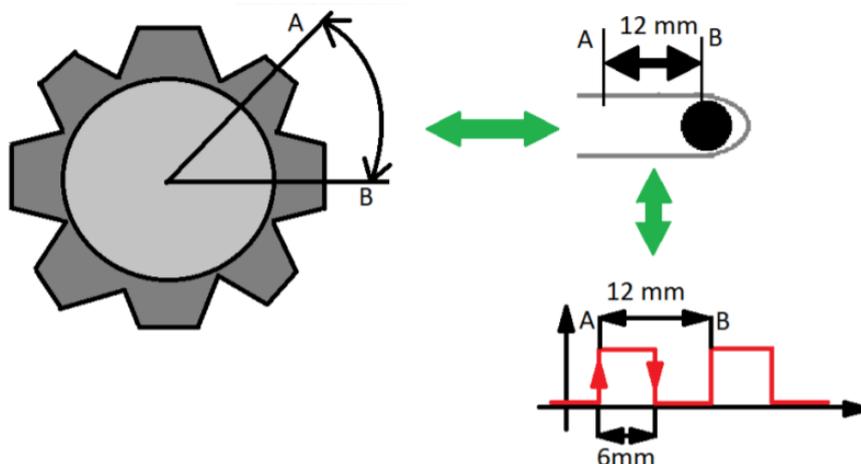


Figure 2.6 Phonic wheel [10]

The following correspondence has been maintained: 1 phonic wheel pulse = 6 mm.

2.1.4 PIVOT

It is the mechanical element which has the task to deliver ranks at the end of the belt: if it remains high, the rank proceeds its route on the distribution, while if it goes down, it delivers the rank to the so-called chart. The pivot is used both for delivering the ranks and for the management of the discarded elements. The pivot extremity is called pen [11].

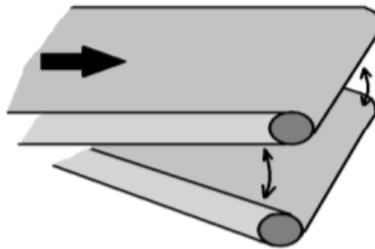


Figure 2.7 Pivot structure [10]

2.1.5 REJECT ZONE

The main reasons that make a rank to be rejected are the following:

- Length: products in the rank are not aligned (detected by FTC1)
- Height: products in the rank are overlapped (detected by FTC2)
- Metal detector: the rank is infected by some external metallic element
- Stop at downstream: machines located downstream are stopped and legs do not accept additional products. Since the source can't be stopped, the ranks are discarded (condition to avoid).

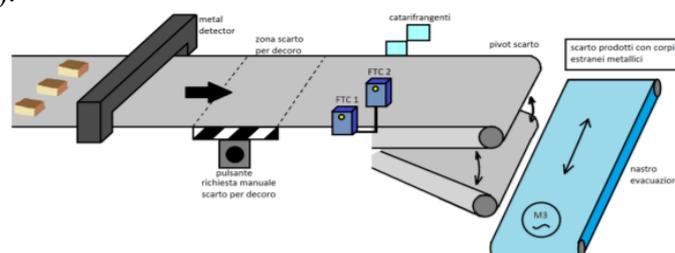


Figure 2.8 Reject zone structure [10]

According to the standard rules, the ranks must be discarded in different directions and according to different solutions. In particular:

- The products discarded by the metal detector must be placed into a locked container;
- The products manually discarded by the operator or for length/height reasons must be placed into an open bucket.

The reject could be performed mainly in two ways:

- Pivot, as shown figure 2.8;
- Pullnose, which acts by means of a mechanical trap, that has a variable opening, given by the distance between the pens of two belts of variable length.

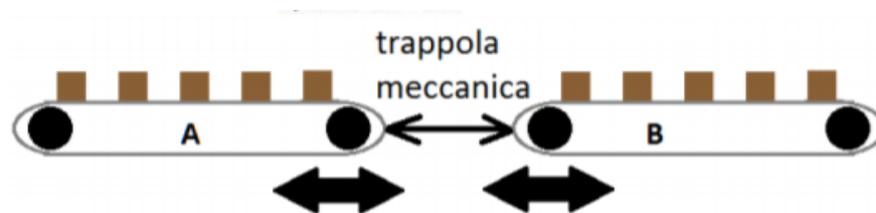


Figure 2.9 Mechanical trap [10]

2.1.6 CHART

For each pivot, a chart is present. The complete configuration of the chart is made up of three motors:

- Entry belt;
- Translator;
- Exit belt (or evacuation belt).

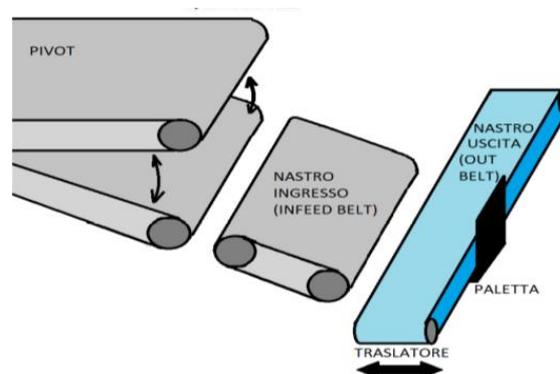


Figure 2.10 Chart configuration [10]

The **entry belt** has the following tasks:

- Receiving the rank;
- Parking the rank;
- Delivering it to the evacuation belt.

The entry belt is characterized by three different velocities, according to the position the rank assumes on the conveyor:

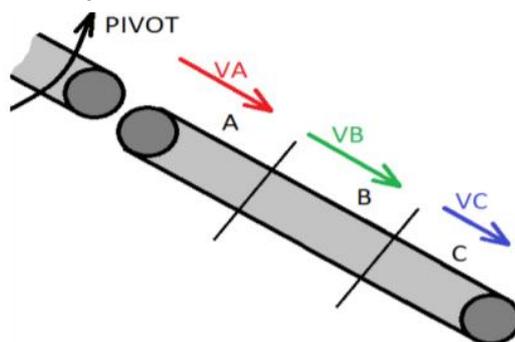


Figure 2.11 Velocities on the entry belt [10]

In particular:

- v_A is called speed of rank reception from pivot;
- v_B is called delivering working speed;
- v_C is called rank storage speed.

The **translator** has the following tasks:

- Sweetly accompanying the product in the passage from entry belt to evacuation belt;
- Place the rank in phase with the turning products zone on the legs.

After the opening of the chart, a recalibration procedure is performed, since it guarantees the redefinition of the zero of the axis after a voltage drop has occurred. The recalibration is automatically performed by the chart, after being reinserted in its place after a cleaning operation. This cleaning operation is automatically performed from the chart, after the chart itself has been opened. The cleaning has the aim of guaranteeing that the chart is free from accidentally fallen products.

The **evacuation belt**, instead, has the following characteristics:

- It takes the product out from the pivot zone, in order to free that zone to receive, then, a new rank;
- It is in gearing with the TT conveyor of the leg;

- It does not perform a gap closing;
- Its length should be as close as possible to one of the ranks. In fact, the ideal working condition is that the head of the rank is as much as possible close to the end of the conveyor, in order to avoid an additional gap given by the difference between the place of delivery of the rank and the end of the belt itself.

2.1.7 CHART WITH DOUBLE PIVOT FOR RECYCLING

With a double pivot solution, it is also possible to use the station both in forward and in backward direction of travel.

The recycling is based on the usage of a storage (a long conveyor, called buffer for simplicity) located at the end of the line: when the ranks are not placed on any leg, for example because the flowpacks (wrapping machines) are stopped, and they continue on the line, in the end they finish on the buffer belt. Then, in a second moment, if there is a lack of product coming from the line, it is possible to feed the legs and the flowpacks with the products stored in the buffer, by acting the recycling modality.

In this way, the last conveyors of the distribution will turn in opposite direction to the forward travelling direction [12].

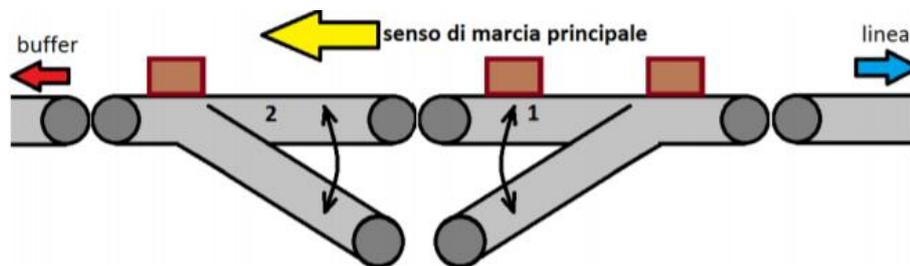


Figure 2.12 Double pivot [12]

In recycling modality, the belt 2 will place the ranks coming from the buffer on the chart. Moreover, it is important that the products are well distanced of a minimum quantity, in such a way that the pivot will be able to go down, deliver the rank on the chart and come back in its high position avoiding that the successive rank is pinched between the pens.

2.1.8 MONITORING AND MANAGEMENT OF RANK POSITIONING

In order to keep track of the ranks positioning on the conveyors of the distribution ridge and then to suitably move the pivot, a technique called “bit shift” is adopted: the portion L of the belt in between the pivot FTC and the belt extremity is taken 500 mm long. By dividing L in segments whose length is equal to 6 mm (pitch of the phonic wheel) and by associating

a bit to each segment in an array of Boolean variables: it is possible to know that at every pulse of the phonic wheel the conveyor moves forward of a distance equal to one step of the phonic wheel (6 mm).

The absence of product on a segment is considered as a 0, while if the product is present on the single segment a bit value equal to 1 is considered.

The pivot FTC sets the bit of the array logic value corresponding to the segment of the belt placed in front of the FTC at each time instant.

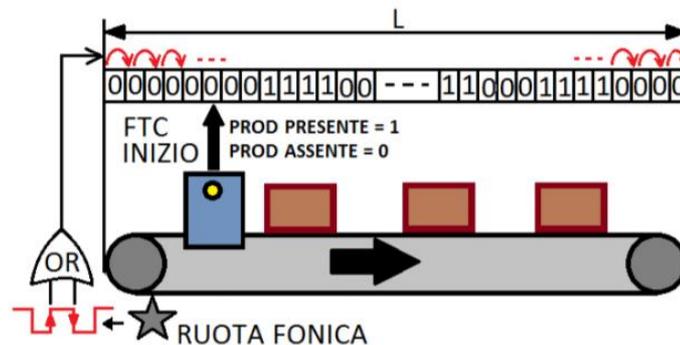


Figure 2.13 Mask for ranks positioning [10]

From the software point of view, this means that a mask of bits is considered and values to these bits are assigned. Furthermore, the mask is shifted forward of a bit for each pulse of the phonic wheel.

With this kind of logic, in order to determine the time at which the pivot should go down, the logic values of the bits of the mask located in a certain interval are checked, where this interval corresponds to the pivot position: if the bits have logic values equal to 1, then the pivot goes down. Starting from the movement the pivot has gone down, a counter is then activated, which considers the pulses of the phonic wheel: after a certain number of pulses, the pivot is placed again in its high position.

2.1.9 PITCH OPERATING MODE

Besides the continuous operating mode, the ridge distribution could also work in the pitch operating mode, so suitably stopping and restarting the belts.

In figure 2.14, the conveyor B assumes the role of a buffer and three types of pitches could be defined:

- Y: rank coming from upstream;
- Z: pitch on FTC begin;

- W: rank to downstream.

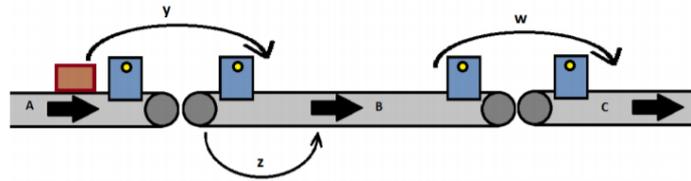


Figure 2.14 Pitch operating mode on the ridge distribution [10]

The conveyor B is initially stopped. When products arrive from the conveyor A, the belt B starts turning and keeps moving until the rank is read by the B begin FTC (pitch y).

B keeps moving since the desired wheelbase between two consecutive ranks is reached (pitch z), then it is stopped.

When B accumulates ranks until its extremity, the conveyor B will start to move again in order to let ranks enter the conveyor C in the correct way (pitch w).

All this procedure is useful both to store ranks on the buffer B, but also to avoid that the ranks lay in the zone located in between the pens of two consecutive belts, since laying in that zone could cause the shredding of the ranks themselves.

2.2 Legs and flowpacks

The legs have the role of receiving the ranks coming from the ridge distribution and move them toward the so-called flowpacks.

In order to work properly, the speed of the legs must go along with the speed of the flowpacks, since if the latter asks for a certain quantity of products, then the leg must be able to feed this need.

The principle adopted is the one of the communicating vessels.

Indeed, the leg can be considered as divided into three different zones:

- **Gap closing zone (or TT zone):** receives the ranks coming from the ridge distribution;
- **Orienting zone:** orients the products;
- **Accumulating zone:** gives the products to the flowpack.

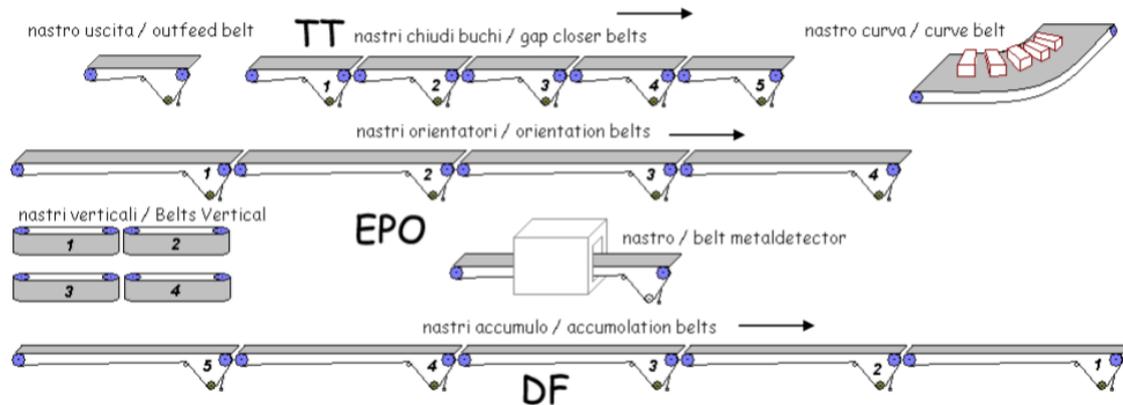


Figure 2.15 Different zones on the leg [13]

2.2.1 TT ZONE

For convention, five different conveyors are taken into account for the gap closing.

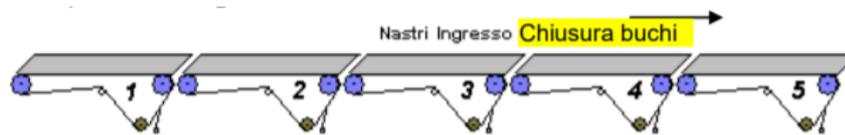


Figure 2.16 TT conveyors [13]

These belts are enumerated from 1 to 5 starting from the arriving direction of the products. They always move in the forward direction.

Usually, the last conveyor of this zone is called “dosing” and it is the master for what concerns the number of products that are going to enter in the flowpack.

Usually, these conveyors must not be too long: in fact, if the conveyor length is greater than the rank’s one, both the FTCs give a 0 level as output and so the conveyor is accelerated to compensate the lack of product and in this way, the conveyors continuously accelerate and decelerate [13].

The last TT, which is located next to the orienting zone, has a reduced role with respect to the previous TT belts. This is due to the fact that a too strong intervention could cause the approaching of the rank to the just distanced product, making null the action of the orienting conveyor. So, this last TT has a hybrid function, that makes it behave more as an orienting conveyor than as a TT.

In this zone the velocity is given by the product of the width of the products and the number of pieces, obtaining in this way the speed in mm/min.

In order to dynamically manage the gap saturation between a rank and another, a correction factor set by the user multiplies the flowpack velocity, separating in this way the conveyor from the wrapper. The gap closing is given by a higher speed step. If the correction

factor is greater than 1, the speed is proportionally increased, while if it is equal to 1, the speed remains unchanged.

The acceleration/deceleration of the different motors is used in order to reach, more or less suddenly, the desired speed step. These acceleration/deceleration ramps will be sweet in the initial zone where the product is not oriented yet, while they will be stronger in the accumulating zone.

In the case of more conveyors dedicated to the gap closing, the velocities are transferred in cascade one conveyor to the other, so that the last belt is the one that follows the wrapper speed, while all the others follow the speed of the preceding one.

2.2.2 ORIENTING ZONE (EPO)

Usually, four different belts are taken into account, even if a smaller number could be sufficient.

The enumeration is progressive from 1 to 4 with respect to the direction of arrival of the products. The direction of motion is always forward.

The ramps of acceleration/deceleration are smooth, since these conveyors must turn the products.

In order to turn the products, two different tapes moving at different velocities are present: in this way, the turning of the products is reached.

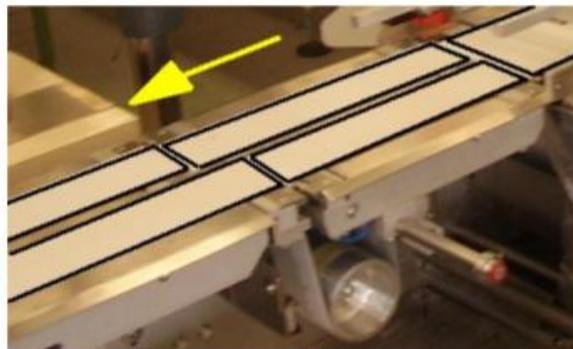


Figure 2.17 Two tapes structure [13]

It is fundamental the separation of the products before the turning action is performed. The jump of velocity needed in order to perform this action is obtained by means of the correction factor, which could be equal or larger than 1: if it is equal to 1, the belts will follow the speed of the flowpack.

These conveyors must not close any gap and so the only solicitations to which they are subject are the ones correlated to the flowpack speed variations.

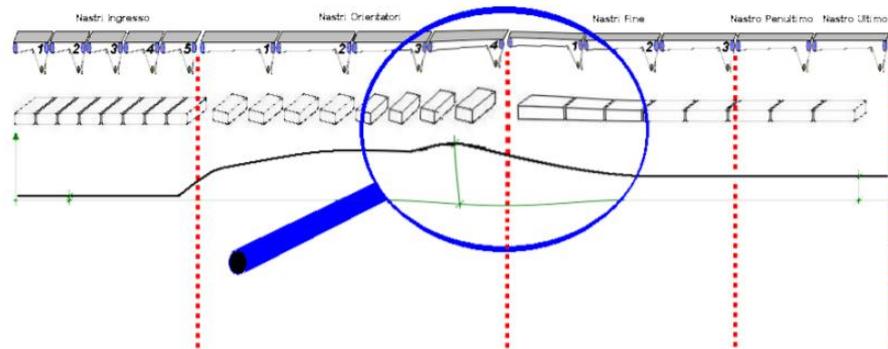


Figure 2.18 EPO zone and products flow [13]

In the circled zone, the metal detector is usually inserted and it is also the zone in which the maximum value of velocity is reached.

The last belt orientator reduces the speed of the products, which has been useful in order to turn the products in the EPO zone. While getting closer to the accumulating zone, the products must be reassembled in order to feed the wrapper.

Different conveyors belong to the orientating zone, in particular:

- The **curved belt** belongs to the EPO zone. This is positioned close to the TT belts and suddenly before the EPO zone. It has only the task of orienting the products and the velocities could be increased while the products pass from this curve.



Figure 2.19 Curve belt [13]

- The **vertical belts** are needed to sweetly orientate the products while travelling on the line. Usually four of these belts are needed in order to suitably accomplish this kind of task. The velocity of these belts are exactly equal to the ones of the horizontal conveyors on which these belts lay.

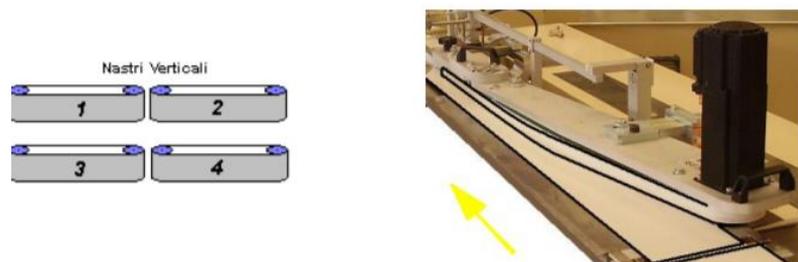


Figure 2.20 Vertical belts [13]

- The **metal detector** is different from a mechanical point of view, but it always belongs to the EPO zone. It is usually located close at the end of the EPO zone and suddenly before the accumulating zone. Its task consists in distancing the products in order to identify the contaminated ones. In this passage, the velocities could be increased till the reading limit of the phonic wheel.

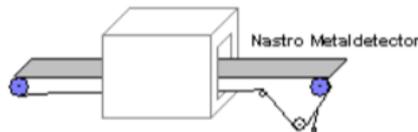


Figure 2.21 Metal detector [13]

2.2.3 ACCUMULATING ZONE

In this zone, five belts are usually used. The larger is the number of belts inserted, the easier will be the ability to close the gaps among the products.

The conveyors are enumerated starting from the zone close to the entrance in the flowpack and their moving direction is always the forward one. In this way, the first accumulator is the one next to the wrapping machine, while the last one is next to the EPO zone.

Usually, the last belt is called “dosing” and it is the master for what concerns the number of products entering the flowpack.

The aim of these belts is to make the flowpack work with no interruptions, by making the last conveyor before the wrapper be never empty.

In this zone the velocity value is computed by knowing the value of the length of the products: the products length multiplied by the number of pieces gives the speed in mm/min.

Also in this zone, the belts have a specific correction factor depending on the possibility to increase or keep constant a conveyor velocity.

In this zone the acceleration/deceleration ramps are more abrupt, since the speeds are higher.

The velocities, as for the other zones, are transferred in cascade among all the belts belonging to this zone.

The different roles of the accumulator belts can be considered and a particular focus is dedicated to the last two belts:

- The **accumulator 2** is always paired to the accumulator 1 and it is highly solicited by accelerations and decelerations.
- The **accumulator 1** precedes the flowpack and so it constitutes the last possibility to close a still existing gap.

2.2.4 SPEED VARIATIONS

During the normal functioning of the legs and the flowpacks a lack of product could occur for one of the following reasons:

- Incomplete ranks;
- Bad oriented products;
- Metal detector discarded products.

In order to manage this lack of product, the speed of every belt is increased thanks to a correction factor, except from the accumulator belts, having in this way more products to be used as a buffer before the wrapper.

When a user defined target value is reached, the increasing of velocity is interrupted and the base velocity is kept.

In this way the system results to be autoregulated, with a hysteresis that can be set according to the user requirements.

The aim of the conveyors is to move at the target speed set upstream, that in absence of variations results to be equal to the speed of the flowpack. If the flowpack slows down, also the belts slow down, except from the ones of the EPO zone, since if the flowpack reduces its speed it is due to a lack of product and so if the belts of the EPO zone follow this variation, the situation would not be solved. For this reason, these kinds of conveyors have the same speed of the flowpack.

2.2.5 OPTIMAL LENGTH OF THE BELTS IN ACCUMULATING ZONE

By considering figure 2.22, it is possible to establish a minimum length of the accumulating belt located just before the flowpack (accumulator 1), in order to let this conveyor not to be empty when the flowpack stops, as:

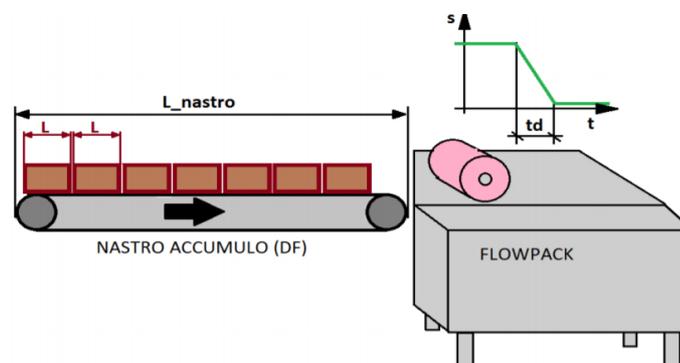


Figure 2.22 Accumulator 1 length [13]

$$L_{belt} = L * s * t_d \quad (2.1)$$

Where:

- t_d is the time needed for the flowpack to stop;
- s is the maximum velocity expressed in hits/min;
- L the product length.

This formula constitutes a necessary condition to guarantee that the accumulator 1 is never empty when the flowpack stops. Anyway, equation 2.1 is an approximation of what really happens and it holds under the assumption that the flowpack always keeps wrapping products at its master speed, then passing from this value of speed to zero. So, an infinite deceleration is considered and this approximation overestimates the number of wrapped products.

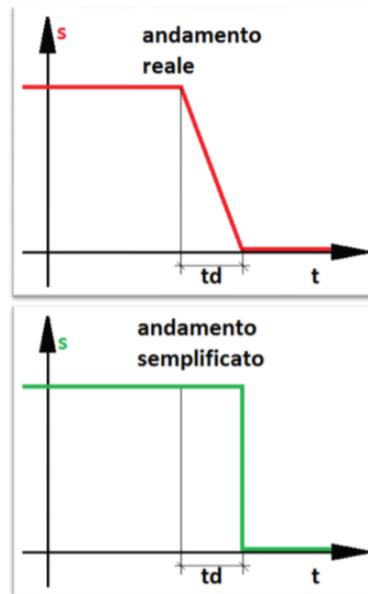


Figure 2.23 Maximum speed graph [13]

2.2.6 SPEED REGULATION IN THE FLOWPACK

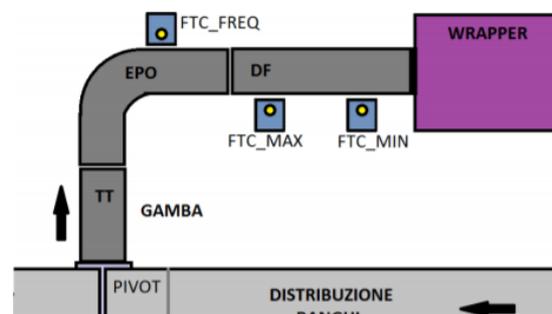


Figure 2.24 Packaging system out from a ridge distribution [13]

The speed imposed to the flowpack could be defined by means of the frequency with which the products arrive to the distribution: this frequency is obtained by measuring the time in between two consecutive products thanks to a FTC located in a specific point of the line. This FTC is indicated as FTC_FREQ in figure 2.24.

In order to manage the speed variations in the flowpack, two FTCs located in the accumulating zone are considered. These are called FTC_MIN and FTC_MAX and they both assume value equal to 1 if the quantity of products is equal or greater than the optimal value. In this situation, the flowpack speed is increased; otherwise, if they both assume value equal to 0, the flowpack speed is decreased since a lack of product is detected. Finally, if the level of product is in between FTC_MIN and FTC_MAX, the flowpack speed is kept equal to the target speed.

2.2.7 CLASSIC SPEED REGULATION

With this kind of regulation, the various elements of the leg are put in gearing with the flowpack: the flowpack acts as the master and it imposes its velocity to all the belts of the leg except from the last accumulating conveyor.

Anyway, this regulation has some limits due to the possible creation of some gaps in the flow of products: in fact, in this case, the conveyors speeds would be reduced, not letting a closing of the gap, since also the flowpack master speed would reduce.

So, a logic that separates the flowpack from the various belts is needed.

2.2.8 SPEED REGULATION WITH COMPENSATION

A possible solution to the creation of the gaps is based on speed compensation. This provides two velocities:

- **Target velocity**, that is the command given to the flowpack;
- **Master speed**, which is the effective flowpack velocity.

The accumulating belts move at the master speed. The EPO belts and the TT ones have a speed regulation based on a feedback coming from the flowpack, defined in this way:

$$v_{TT} = v_{EPO} = |(v_{target} - v_{wrp}) + v_{wrp}| = |v_{diff} + v_{wrp}| \quad (2.2)$$

Where:

- v_{TT} is the velocity of the TT belt;
- v_{EPO} is the velocity of the EPO belt;
- v_{target} is the velocity to be achieved;

In moving backwards the logics of the conveyors are the opposite of the ones in forward moving direction: when the belts move in backwards, it is fundamental that the ranks are well separated one from the other, since there will be surely a point in which the pivot must go down and if the procedure cannot be suitably performed, an operator must intervene to re-establish the correct functioning of the line [14].

In backwards moving direction, a manoeuvre called approaching is used, which is shown in figure 2.26.

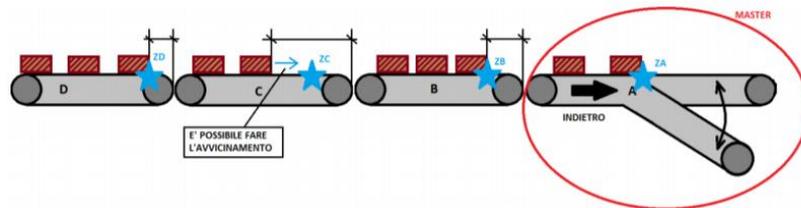


Figure 2.26 Approaching maneuver [12]

The conveyor C does not have the ranks in the point z_C . The points z_A , z_B , z_C , z_D represent the end of the free space and the beginning of the successive belt, so the exact point where the rank is going to be stationed. When the conveyor A, which is the master for all the following belts B, C, D, starts recirculating and delivers its rank either to the chart or to the remaining upstream portion of the line, then all the other conveyors start moving with it. Instead, when A stops, all the other belts keep moving since the ranks reach the respective z positions. In this way it is possible to have all the ranks at the desired distance one from the other.

2.3.1 DOUBLE RECYCLING

Usually, when a line is provided with a recycling pivot, the last legs are used as jolly legs, so they are only useful to empty the buffer in the shortest time possible, in order to guarantee a suitable available capacity to be filled.

The types of double recycling taken into account are essentially two: not simultaneous and simultaneous:

- The **not simultaneous double recycling** takes place only in presence of two legs used for the recycling, when the last one cannot receive products because a failure has occurred.

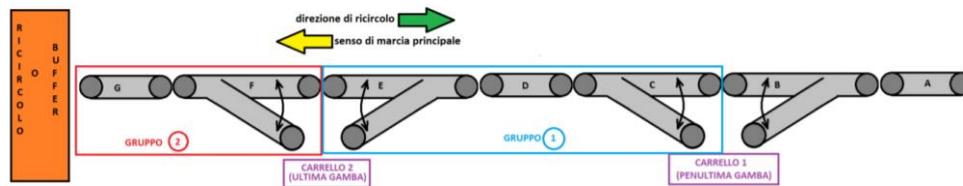


Figure 2.27 Double recycling structure [12]

The aim of this not simultaneous double recycling is to move the ranks coming from the recycling, previously directed towards the chart 2, to the group 1. The crucial point is in between the belts F and E.

It is important to not stop the products in between the belts F and E, because when the chart 2 restarts, an inversion of motion is needed to eliminate the rank that does not allow the pivot to go down, as shown in figure 2.28.

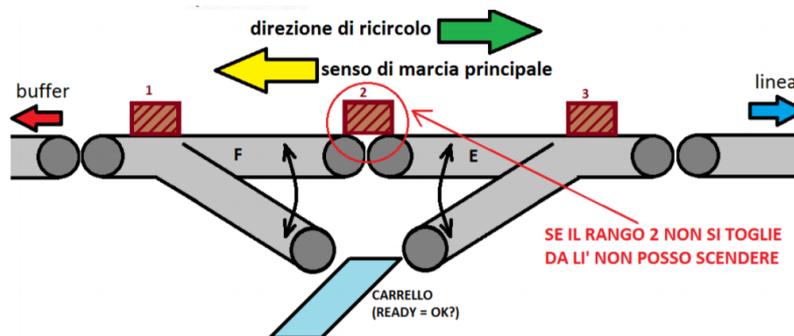


Figure 2.28 Blocked situation with the rank stuck [12]

The only solution consists in moving forward both E and F and so all the connected belts (G, C, D), since some ranks could be stuck in between these conveyors.

In this way, the situation could lead to the rank 3 delivered to the chart 2, the rank 1 that has overpassed the conveyor F and the rank 2 ready to be recycled.

The limit of this procedure is that it is really time consuming, also risking that the legs are not uniformly fed.

In order to avoid this situation, the procedure shown in figure 2.29 is adopted.

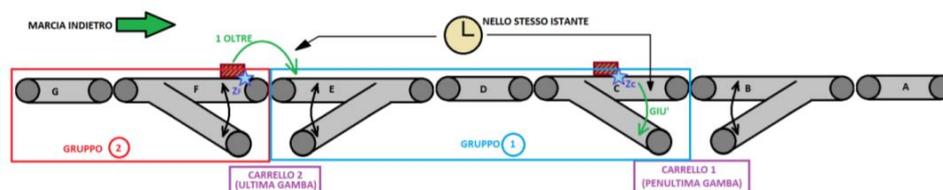


Figure 2.29 Solution for the not simultaneous double recycling [12]

By knowing the mechanical symmetry of the considered line, the rank in z_C will surely travel more or the same distance than the rank located in z_F in order to jump on the belt E.

So F is going to move only when also C starts moving and in this way it is guaranteed that no rank is going to be stuck in between the conveyors F and E.

- The **simultaneous double recycling** works exactly like the not simultaneous, with the difference that two legs are present.

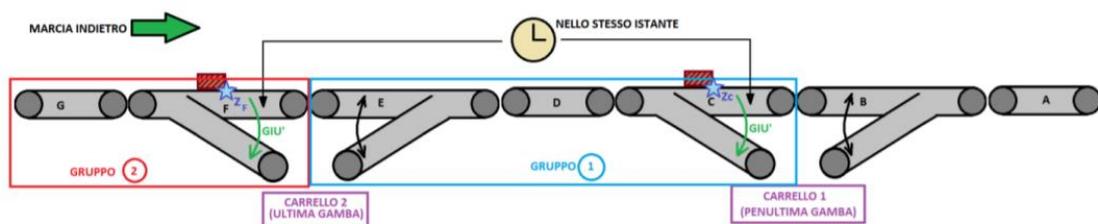


Figure 2.30 Simultaneous double recycling [12]

In this case, based on the simultaneous starting of the conveyors C and F, the two charts are going to be fed.

The issue is to correctly feed the legs: in order to do this in the best way, it is important to maintain the ranks on G as close as possible among them, aiming at always having the availability of some ranks to be delivered to the legs.

In real working conditions it could happen that a rank gets stuck in between the belts F and E and the adopted solution consists in keeping the conveyors moving in backwards:

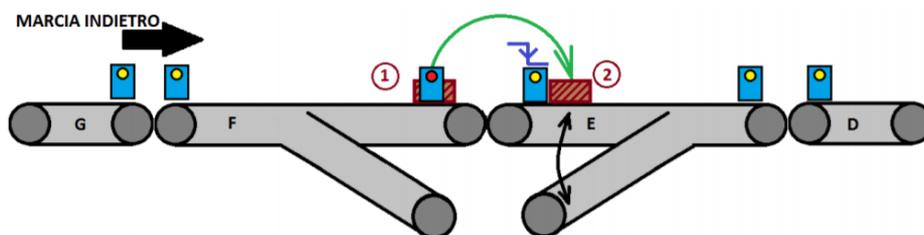


Figure 2.31 Alternative solution to avoid products in between two pens [12]

This technique essentially consists in keeping moving in backwards since both the FTCs are overpassed. In particular, the situation of figure 2.32 is considered.

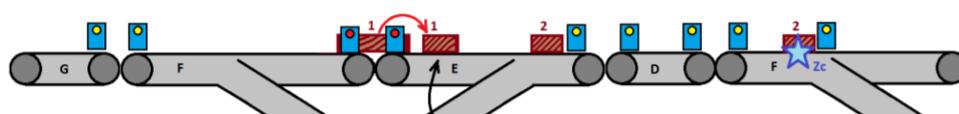


Figure 2.32 Chained procedure to free the pens [12]

In order to release the initial FTC of the conveyor F and the FTC located at the end of the conveyor E, both E and F move in backwards. If the rank 2 located on E does not reach the FTC at the beginning of E, nothing happens. Instead, if the rank 2 obscures the FTC it means that it is going to be placed in between the pens and this gives the start to the backward movement of the belt D. The chain stops when the rank reaches the position z_C [15].

Chapter 3 - The concept of Class

This chapter is devoted to the strategies adopted in order in the software implementation of the models of the main objects composing the two reference distributions plants.

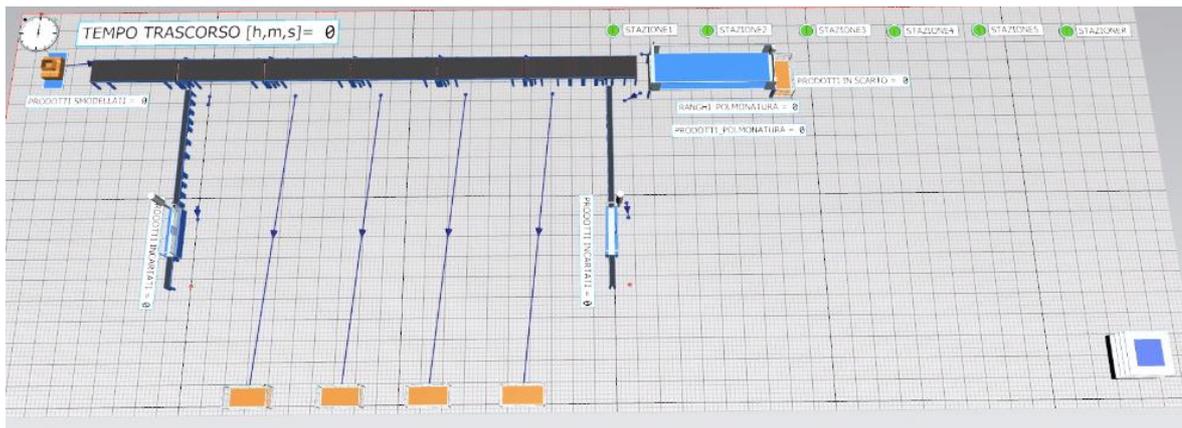


Figure 3.1 Software representation of the plant with fan buffer.

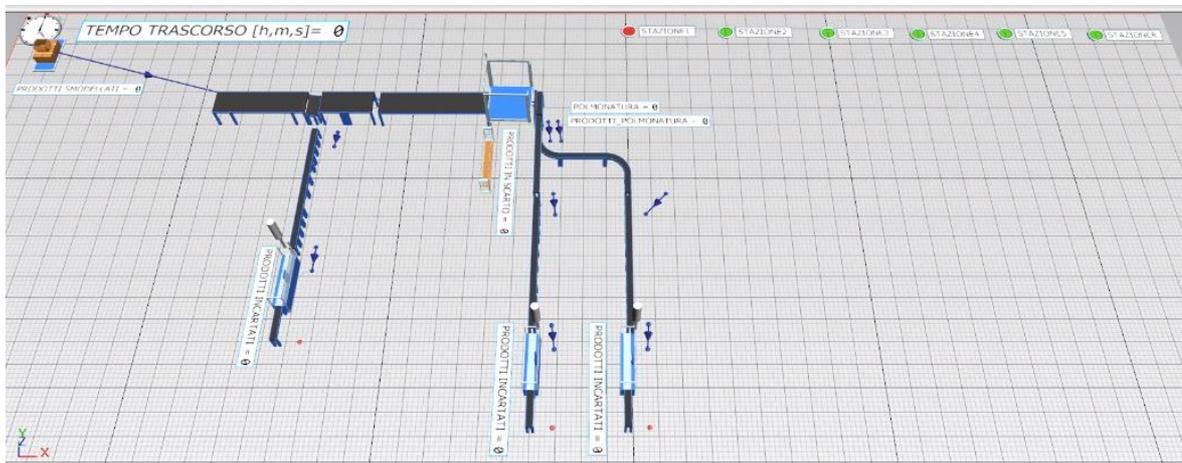


Figure 3.2 Software representation of the plant with gondola buffer.

Basically, being Tecnomatix Plant Simulation a software tool based on an object oriented programming language, the different models have been implemented as classes. A class is a data structure that is composed by two important elements, called Attributes and Methods.

The Attributes constitute the set of variables that are needed to characterize the object that the class represents. Attributes can be given by either physical quantities or logical quantities and, depending on the particular element they represent, their values can be integer, real or Boolean. A typical example of a real attribute is the “speed” of a Pivot, an example of an integer attribute is the “capacity” of a Buffer, while an example of a Boolean attribute is the “empty” attribute of whatever machine in the production line.

The Methods constitutes the a set of functions that regulates the behavior of the class; basically, on the bases of the sensing and the evaluations through dedicated algorithms of the values of specific attributes of interest, the methods provide the actuations by setting the values of other attributes. Being the software platform event-based, the methods are executed by the processor at specific instants, according to specific events that are triggered by the program; the main kinds of methods that have been used in order to implement the behavior of the different developed objects are illustrated here after.

The “OnEntrance” method is a function that is executed each time a unit equipment enters the object represented by the class, the “OnExit” method is a function that is executed each time a unit equipment exits the object represented by the class; notice that these methods are usually called very often, especially in applications belonging to the food domain, in which the number of produced units equipment is huge, and so they need to present a low computational complexity, avoiding non-deterministic loops, otherwise the processor will become easily congested.

The “OnBackwardEntrance” and “OnBackwardExit” methods are functions called when a unit equipment enters the nominal exit of the object and, vice versa, exits from the nominal entrance of the object respectively; these methods are mainly used in order to implement recirculating tasks and, hopefully, they are called with a lower frequency.

The “OnSensor” method is a function called whenever a unit equipment reaches a specific point in the space of a certain object; for the addressed applications, the pieces most of the time present a regular shape with a well-defined length and so the sensing of the piece can be further divided and performed on the front or on the rear of the piece.

The “OnObserver” method relies on an event affecting an attribute of interest; indeed, it is a function that is called whenever a certain variable reaches a specific value or, sometimes, simply when such variable changes its value.

Finally, three methods have been defined, called “Reset”, “Init” and “EndSim”; they are standard methods, in the sense that they are in common to all the created classed, so each class presents them, and they are called at different points in time. The “Reset” and “Init” methods are functions called before the beginning of the actual simulation; they are mainly intended for initialization purposes. In particular, the “Reset” method is needed for the assignment of the default values to predefined variables, like the starting value of a counter. As will be illustrated, the “Init” method is used for handling the management of the input parameters, setting the parametrized variables to the values selected by the operators in order

to perform the simulation; an example of these variables is the simulation time, which is totally configurable by the operator, on the basis of his needs. On the other hand, the “EndSim” method is a function that is called when the simulation time is elapsed; as we will see, it is responsible for the evaluation of the performance of the distribution and for the generation of the charts representing statistics of the metrics of interest.

Two other important concepts are inheritance and visibility. As already discussed, each object created through class is characterized by its own specific attributes and methods, and it represents a sort of reference model; then, according to the plant that has to be represented, different instances of that reference object are brought into the frame. According to the inheritance mechanism, each instance will inherit the characteristics of its reference class, presenting the same list of attributes and methods. Then, according to the visibility feature, some methods and attributes can be public and so they can be tested and utilized by other objects belonging to other classes, or they can be private and so they can be exploited only by the class to which they belong. In the developed project, most of the time the methods have been implemented as private, while the majority of the attributes are public, since the behavior of a machine is often based on the evaluation of some parameters of its succeeding or preceding.

A final consideration must be done on a simplification that the software tools offers. Tecnomatix Plant Simulation provides three main predefined building blocks, that are the conveyor block, the buffer block and the station block. These blocks represent the three main families to which a certain machine can belong to and they are “virgin”, in the sense that they are equipped with their basic functionalities; then, it is up to the user to add and define more complex and specific functionalities through the definition of dedicated attributes and methods. In particular, the basic functionality of the conveyor block is to carry the product all over the distribution plant, the basic functionality of the buffer block is to store the product making it available for recycling and the main functionality of the station block is to machine the product performing a certain action on it.

Notice that, according to the company’s specifications, a lot of graphical simplifications have not been performed; in this perspective, most of the time, the main objects have been created and modeled maintaining the components of their real structures. This choice is dictated by the will of fulfilling the needs from a client’s perspective; anyway, since each different component could be equipped with different methods, this solution adds complexity from a computational point of view.

In the following, the classes of the main objects composing the two reference distributions plants will be introduced, focusing in particular on the different methods and attributes defined in order to perform and represent via software the actual behavior of those components.

3.1 Class “PIVOTSX”

The PIVOTSX class represents one of the main objects composing the ridge distribution, through which the molds flow. According to its real structure, its implementation has been performed exploiting three segments, represented by three different powered conveyor blocks.

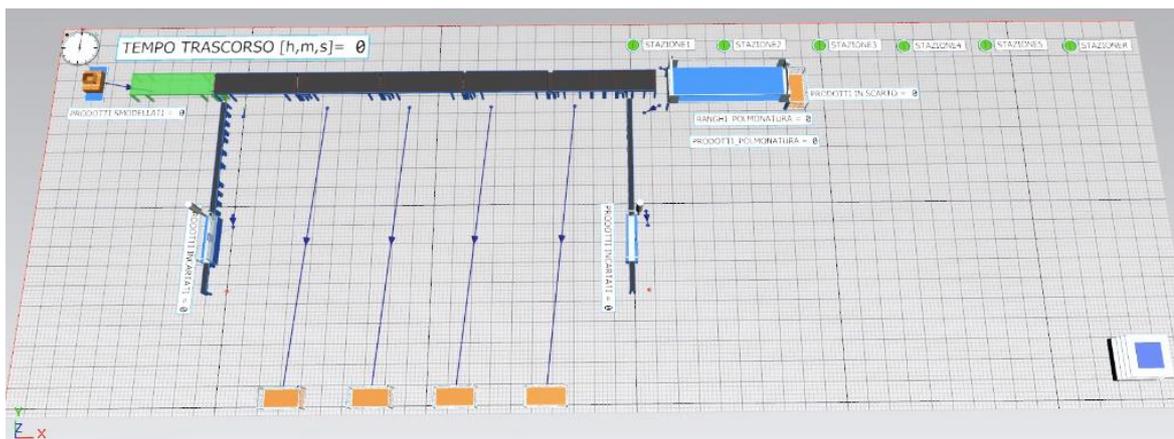


Figure 3.3 PIVOTSX model in the whole distribution plant.

The first conveyor has been denoted as D_P_X, that stands for DORSALE_PIVOTTANTE. It represents the main conveyor of the ridge distribution, the one that has the capability of pivoting feeding the inclined conveyor, if the leg is ready to receive the product, or staying up feeding the next D_P_X conveyor of its following PIVOTSX object. As will be discussed in more detail, this first block is characterized by two methods, that are the “OnSensor” method and one “OnExit” method.

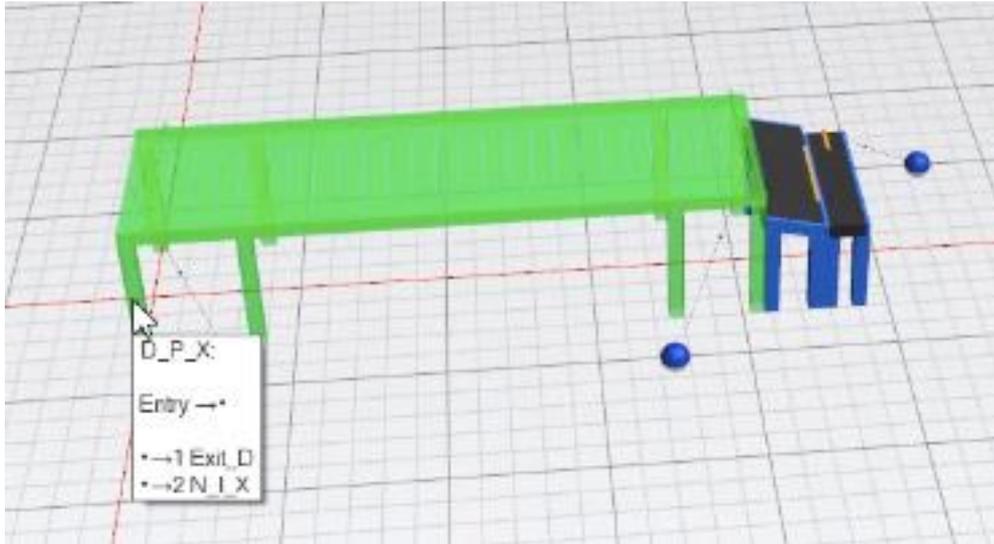


Figure 3.4 D_P_X conveyor of PIVOTSX model.

The second conveyor has been denoted as N_I_X, that stands for NASTRO_INCLINATO. It represents the so-called “infeed belt”, the one that has the capability of receiving the rank from the pivot and delivering it to the evacuation belt. As will be discussed in more detail, this second block is characterized by one method, that is the “OnExit” method.

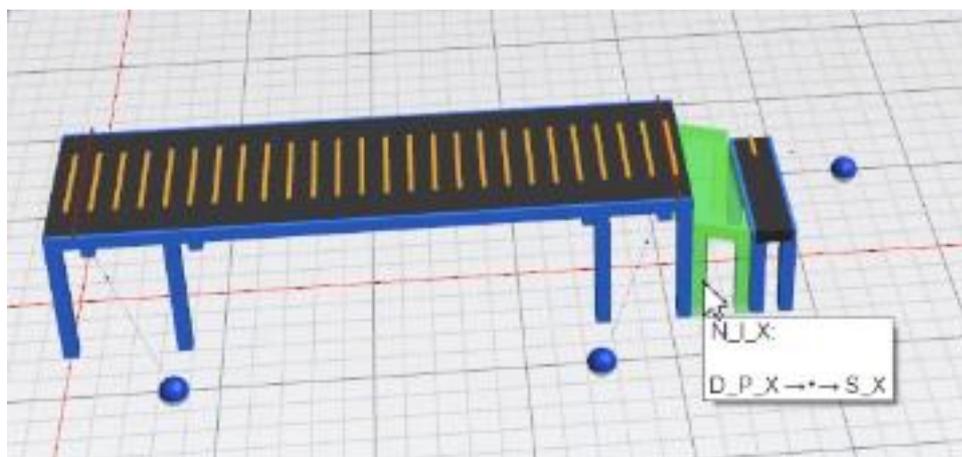


Figure 3.5 N_I_X conveyor of PIVOTSX model.

The third conveyor has been denoted as S_X, that stands for STRAPPATORE. It represents the so-called “evacuation belt”, whose task is taking the product out from the pivot zone and feeding its following leg. This third block is not characterized by any method; this means that it simply receives the product and carry it at constant nominal speed, without implementing any particular logic.

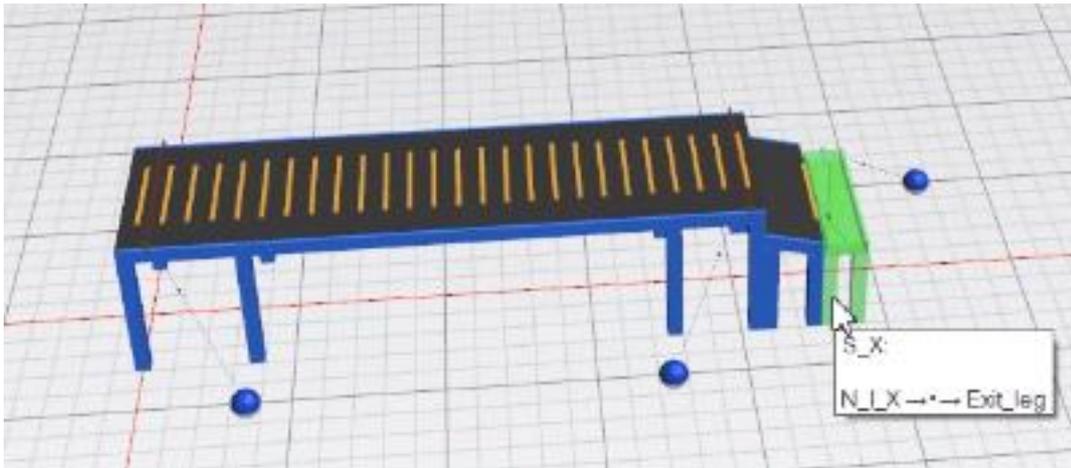


Figure 3.6 S_X conveyor of PIVOTSX model.

In the following, the methods of the different blocks and the attributes that they exploit to implement the logic will be presented and discussed.

3.1.1 “D_P_X” METHODS

As already said, the methods of the D_P_X conveyor are one “OnSensor” method and the “OnExit” method.

The “OnSensor” method is executed every time the piece passes a virtual sensor placed at 100 mm from the end of the conveyor; in particular, the function is called whenever the front of a product passes through the sensor. The program implemented by the “OnSensor” method acts on a user-defined attribute, called “pivot_basso”. This is a Boolean variable, initialized as false, that is needed in order to identify the pivoting of the D_P_X conveyor; when it is true it means that the conveyor has performed the pivoting, when it is false it means that the conveyor didn’t perform the pivoting. For what concerns the logic of the “OnSensor” method, it is based on a simple if-then-else implementation. In particular, if the Boolean attribute “empty” of the conveyor N_I_X is sensed as true, the pivoting is performed by setting the graphical attribute “setCurveSegments” of the conveyor to -0.2

meters and the variable “pivot_basso” is set true. On the other hand, if the N_I_X’s attribute “empty” is sensed as false, the variable “pivot_basso” remains false.

The “OnExit” method evaluates the attribute “pivot_basso”, again according to an if-then-else structure. In particular, if “pivot_basso” is sensed as true, then the piece is moved to the N_I_X conveyor through the command “move” and the segment of the conveyor is realigned to its starting horizontal position; finally, the value of “pivot_basso” is set again to false, ready to be eventually modified when the following piece will pass through the sensor. On the other hand, if “pivot_basso” is sensed as false, the piece is moved to the D_P_X conveyor of the following PIVOTSX composing the ridge distribution.

Notice that the logic implemented by the “OnSensor” method is a sort of decision logic, on the bases of what the succeeding “OnExit” method knows about where the piece has to be moved; this is done in order to implement a behavior as similar as possible to the real one, in which the machine knows at the sensor level, through an encoder, if the pivoting must be performed.

3.1.2 “NI_X” METHODS AND ATTRIBUTES

As already said, the only method of the N_I_X conveyor is the “OnExit” one.

The “OnExit” method has the functionality of implementing the passage of the different ranks composing the mold from the N_I_X conveyor to the S_X conveyor. This operation is performed in a purely graphical way, exploiting two variables called “RM_NumeroRanghi” and “PR_NumeroProdotti”; they are user-defined variables, the first represents the number of ranks per mold, the second is the number of products per rank. According to this graphical approach, first the rank that is going to exit the N_I_X conveyor and to enter the S_X conveyor is deleted through the “deleteobject” function. Then a double for-cycle is performed, in order to place on the S_X conveyor the different products composing the previously deleted rank; the first for-cycle is performed on the variable “RM_NumeroRanghi” and it waits until the successive conveyor is empty, the second for-cycle is performed on the variable “PR_NumeroProdotti” and it is responsible for the placement of the different single products on the successive conveyor. In particular, this is done exploiting the “create” function, through which it is possible to place a new part in a specific position of the conveyor; in this way, it has been possible to place the different products on the S_X conveyor in a precise manner, according to the suitable gap between them.

Notice that, after this operation, the unit equipment, i.e. the basic part that travels on the conveyors, switches from the rank to the piece, that is the actual product.

3.2 Class “PIVOTDX”

The PIVOTDX class represents one of the main objects composing the final part of the ridge distribution. According to its real structure, its implementation has been performed exploiting five segments, represented by five different powered conveyor blocks. Notice that three of these five conveyors are the ones composing the PIVOTDX class, with some further modifications on the logic implemented by the methods.

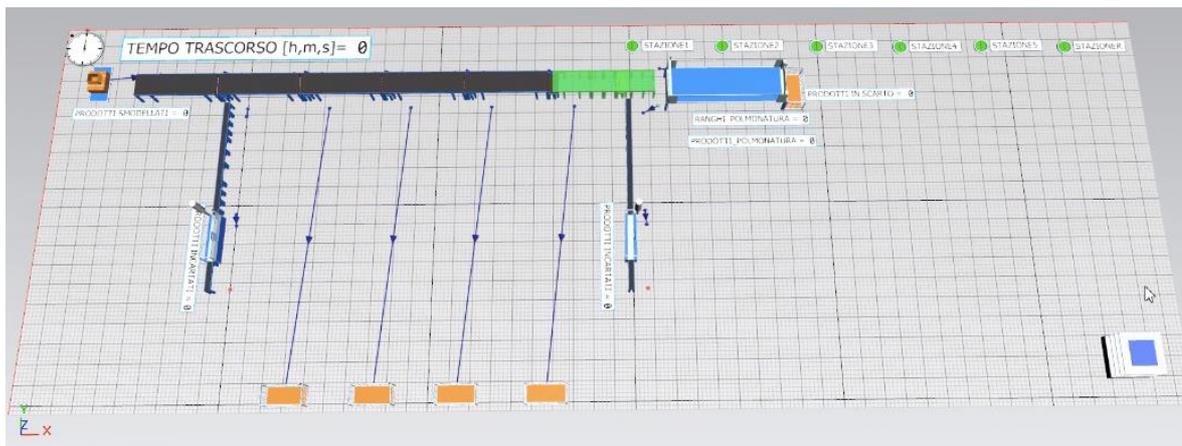


Figure 3.7 PIVOTDX in the whole distribution plant.

The first conveyor has been denoted as D_X, that stands for DORSALE. It represents an element of the main ridge and it has not some complex logic to implement; for this reason, it is not characterized by any specific method, but it simply takes the products towards the subsequent conveyor.

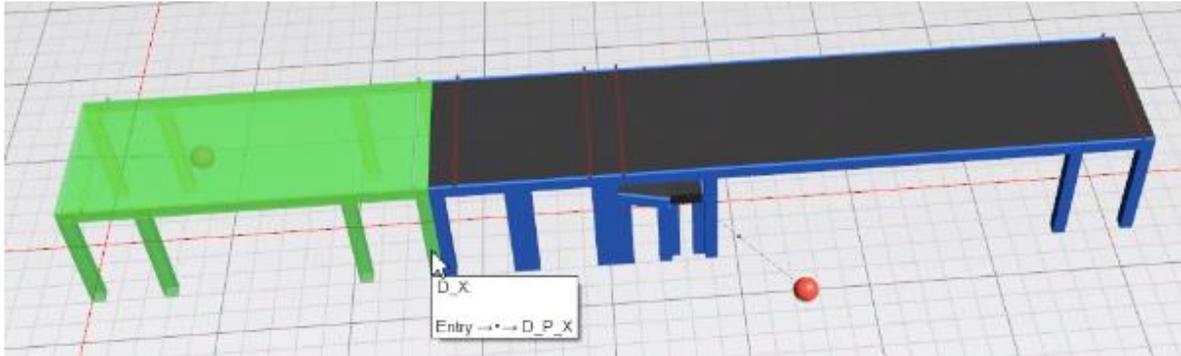


Figure 3.8 D_X conveyor of PIVOTDX model.

As anticipated, the second, third and fourth conveyors are the ones that compose the PIVOTDX object. In this case, the names of the conveyors remain the same and also the kind of methods by which they are characterized; in this perspective, the conveyor D_P_X is characterized by the “OnSensor” and “OnExit” methods, while the conveyor N_I_X is characterized by the “OnExit” method. For what concerns the logics, the programs are essentially the same for the flow of the products in the nominal direction, while the methods of the D_P_X conveyor are characterized by some additional instructions needed in order to manage the flow of the products in the backward direction and so concerning the recycling issue.

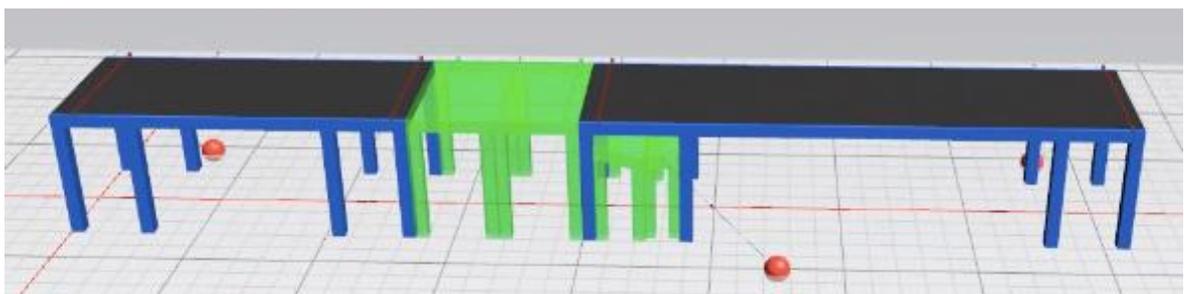


Figure 3.9 D_P_X, N_I_X and R_X conveyors of PIVOTDX model.

The last conveyor has been denoted as D_R_X, that stands for DORSALE_RICIRCOLO. It is the element preceding the Fan buffer in the first distribution and it implements the actual logic for managing the recycling. To do that, it is equipped with two sensors, each one

characterized by an “OnSensor” method; moreover, it presents the “OnEntrance”, the “OnExit” and the “OnBackwardExit” methods.

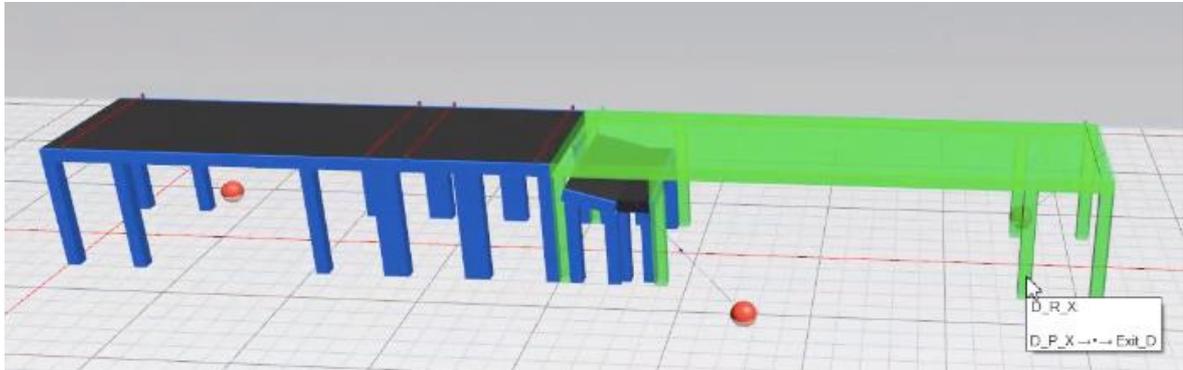


Figure 3.10 D_R_X conveyor of PIVOTDX model.

In the following, the methods of these different blocks and the attributes that they exploit to implement the logic will be presented and discussed.

3.2.1 “D_P_X” METHODS

The kinds of methods characterizing the D_P_X and N_I_X conveyors are essentially the same discussed for the implementation of the PIVOTSX class, with some additional elements for the methods of the D_P_X conveyor, in order to manage the recycling phase.

In this perspective, for what concerns the “OnExit” method of the D_P_X, an additional set of instructions has been implemented. If the variable “pivot_basso” is sensed as false then some actions are performed on the succeeding D_R_X conveyor, before moving the part on it through the command “move”; in particular, its Boolean attribute “Backwards” is set to false and its attribute “Speed” is set to the nominal value of 0.4 m/s.

Notice that these instructions are fundamental in order to avoid a mismatch between the directions of flow of the D_P_X and D_R_X conveyors; indeed, the D_R_X conveyor could have been previously used for the recycling and so travelling towards the backward direction.

3.2.2 “D_R_X” METHODS

As already mentioned, the D_R_X conveyor is equipped with different methods, needed to manage the operations concerning both the nominal flowing direction of products and the backward direction for the recycling. Let’s analyze these methods, presenting the main

functionalities implemented by them; notice that the order in which the methods are illustrated is coherent with the actual order that the product encounters while flowing on the conveyor.

The “OnEntrance” method performs some basic actions in order to properly configure the main parameters of the D_R_X conveyor when a product is going to enter the conveyor. In particular, it sets its “Backwards” attribute to false and its attribute “Speed” to the nominal value of 0.4 m/s. These operations are needed since the D_R_X conveyor is used also for the recycling issue and for the step pace issue, and then the “Backwards” and “Speed” attributes may have been modified by other methods. Moreover, it sets the initial value of a Boolean user-defined attribute “pivotbasso_r” equal to false. The role of the “pivotbasso_r” variable is conceptually the same of the “pivot_basso” variable for the PIVOTSX class; it is a Boolean variable used to identify the pivoting of the D_R_X conveyor in the backward direction; when it is true it means that the conveyor has performed the pivoting, it is false otherwise.

The first sensor is placed at 150 mm from the beginning of the conveyor and it is sensible to both the front and the rear of the product; its implemented “OnSensor” method performs some more complex actions, managing some issues related to the nominal and the backward directions. For what concerns the nominal direction, the main logic is the one implementing the step pace, which lets a product entering the D_R_X conveyor remain still at the beginning of the conveyor, waiting to be recycled, advancing one step forward only if another product enters the conveyor. In this way, the time needed to perform the recycling through the backward pivoting is smaller, since the product is ready to be pivoted. The step pace is performed by setting the “Speed” attribute of the D_R_X conveyor to 0 and its “Backward” attribute to true, being ready for an eventual recycling of the stopped product; then the logics for managing the arrival of a further product, and the advancement of one step, are implemented by the “OnEntrance” method, setting “Backward” to false and “Speed” to the nominal value. For what concerns the backward direction, the main logic consists in the implementation of the pivoting for the recycling: when the last entered product is still at the beginning of the conveyor, the sensor checks whether it can be recycled and so whether the backward pivoting can be performed. This operation has been implemented via software through a “waituntil” instruction, that simply waits until the preceding NI_X and D_P_X conveyors are both empty; if this condition is verified, then the pivoting is performed by setting the graphical attribute “setCurveSegments” of the conveyor to -0.2 meters and the variable “pivotbasso_r” is set true.

The “OnBackwardExit” method evaluates the “pivotbasso_r” attribute through an if-then command. In particular, if “pivotbasso_r” is detected as true, then the piece is moved to the NI_X conveyor through the command “insert” and the segment of the conveyor is realigned to its starting horizontal position; finally, the value of “pivotbasso_r” is set again to false. Notice that the idea behind the combination of the first “OnSensor” method and the

“OnBackwardExit” method is actually very similar to the one implemented by the PIVOTSX object in order to perform the pivoting in the nominal direction; in this perspective, the “OnSensor” implements a sort of decision logic, on the basis of which the subsequent “OnBackwardExit” method moves the product for the recycling.

The second sensor is placed at 100 mm from the end of the conveyor and it is sensible to the front of the product; its implemented “OnSensor” method performs some simple operations, needed to avoid mismatch between its flowing direction and the flowing direction of its subsequent element, which is a conveyor belonging to the buffer object. Indeed, the piece is going to be moved to the subsequent conveyor that, being used also for recycling, could have the “Backwards” attribute still set to true; for this reason, by exploiting the “Succ” attribute, the “Backwards” attribute of the succeeding conveyor is set to false and its “Speed” attribute is set to 0.4 m/s.

Finally, the same operations are performed also by the “OnExit” method, before moving the piece to the buffer through the command “move”.

3.3 Class “LEGX”

The LEGX class represents the series of components through which the single ranks flow, after being unloaded from the single pivot of the ridge distribution. According to its real structure, its implementation has been performed exploiting eleven powered conveyor blocks, composing the three different zones of the leg distribution.

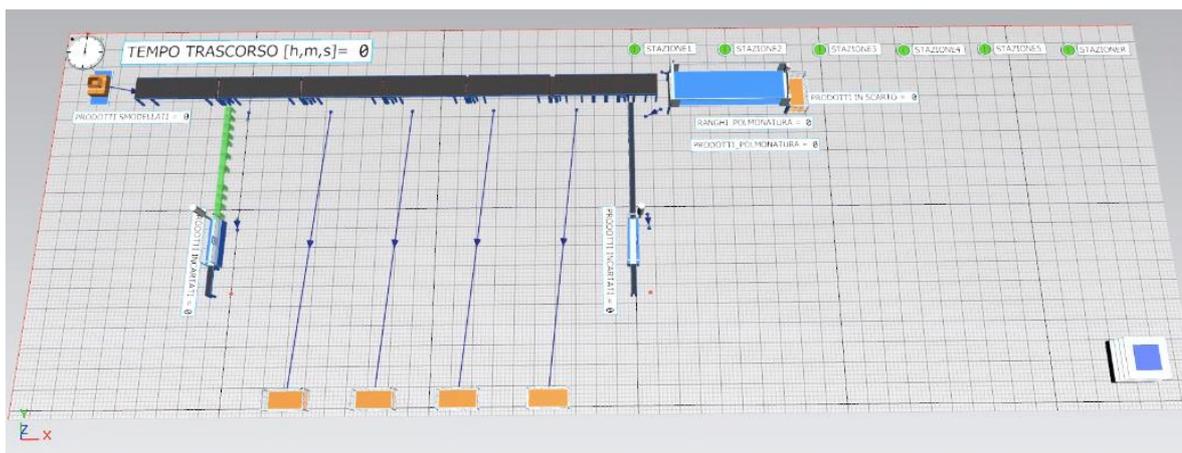


Figure 3.11 LEGX model in the whole distribution plant.

The first, second and third conveyors, denoted as G_TT1_X, G_TT2_X and G_TT3_X, represent the Gap closing zone (or TT zone).

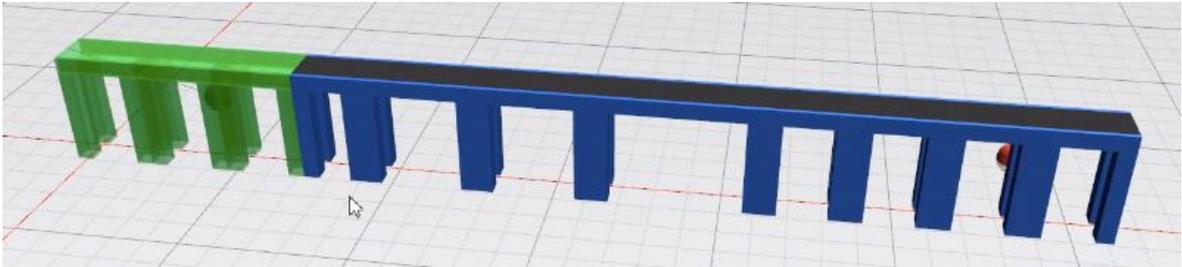


Figure 3.12 G_TT1_X, G_TT2_X and G_TT3_X conveyors of LEGX model.

The fourth, fifth and sixth conveyors, denoted as G_OR1_X, G_OR2_X, G_OR3_X, represent the initial part of the Orienting zone (or EPO zone). This part is responsible for the positioning of the products rotating them from a horizontal to a vertical position.

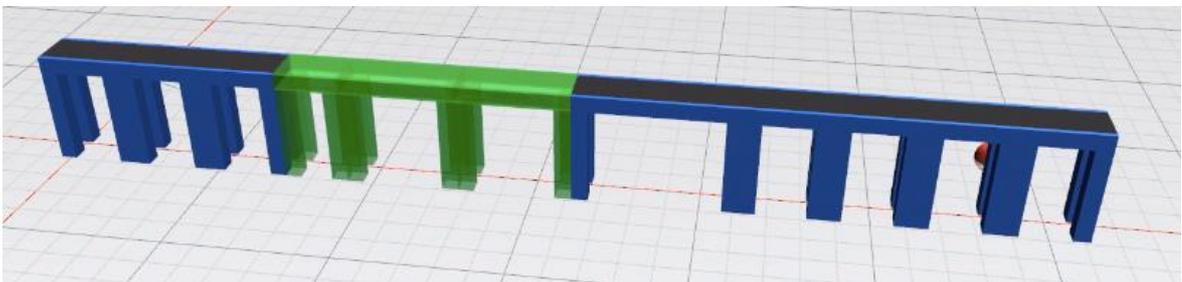


Figure 3.13 G_OR1_X, G_OR2_X and G_OR3_X conveyors of LEGX model.

The seventh conveyor, denoted as G_METAL_X, represents the final part of the Orienting zone (or EPO zone). This part is responsible for checking the cleanness of the products, revealing if some metallic residues contaminate them.

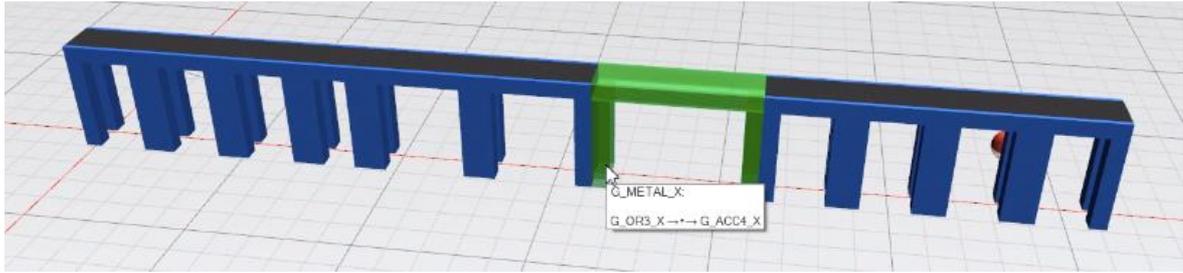


Figure 3.14 G_METAL_X conveyor of LEGX model.

The last four conveyors, denoted as G_ACC4_X, G_ACC3_X, G_ACC2_X, and G_ACC1_X, represent the Accumulating zone.

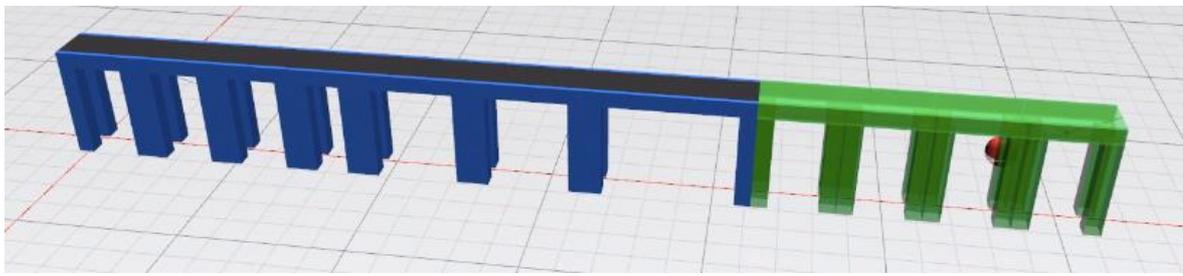


Figure 3.15 G_ACC4_X, G_ACC3_X, G_ACC2_X and G_ACC1_X conveyors of LEGX model.

The speeds of the different elements of the leg are regulated by the processing time of the Flowpack that follows them, and are configured according to a specific function called “UpdateSpeed”, as discussed in the sub-section about the management of the input parameters.

Apart from that, the methods characterizing the LEGX class are those implementing the rotation of the products in their passage through the Orienting zone; in particular, this is performed exploiting the “OnEntrance” method of the OR_1_X and OR_2_X components, as discussed here after.

3.3.1 “OR_1” AND “OR_2” METHODS

The rotation of the products by 90° in their passage through the Orienting zone has been performed exploiting a graphical function called “objectangle”, that allows to rotate the position of the unit equipment of a certain angle about a given axis.

The rotation has been performed gradually, as it happens in reality in order to avoid damaging the products. In this sense, the “OnEntrance” method of the OR_1_X conveyor performs a rotation of 30° of the product; then, the “OnEntrance” method of the OR_2_X conveyor performs a further rotation of 60°, thus obtaining a complete rotation of 90°.

3.4 Class “FLOWPACKX”

The FLOWPACKX class represents the object following the leg distribution, the one that is responsible for the machining of the product. According to its real structure, its implementation has been performed exploiting two different blocks.

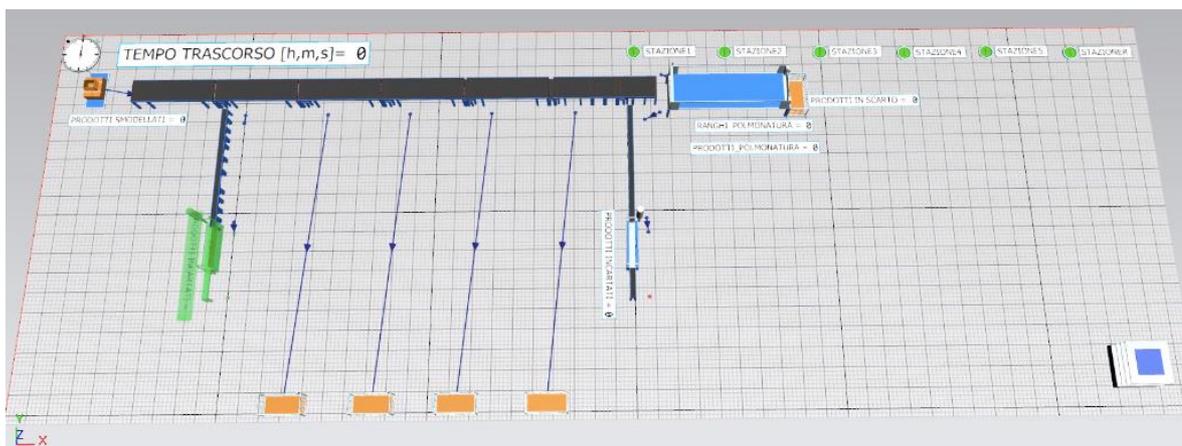


Figure 3.16 FLOWPACKX model in the whole distribution plant.

The first block, denoted as FLOWPACK_X, belongs to the standard station component provided by the software, and it is the one responsible for the actual flow packing of the products.

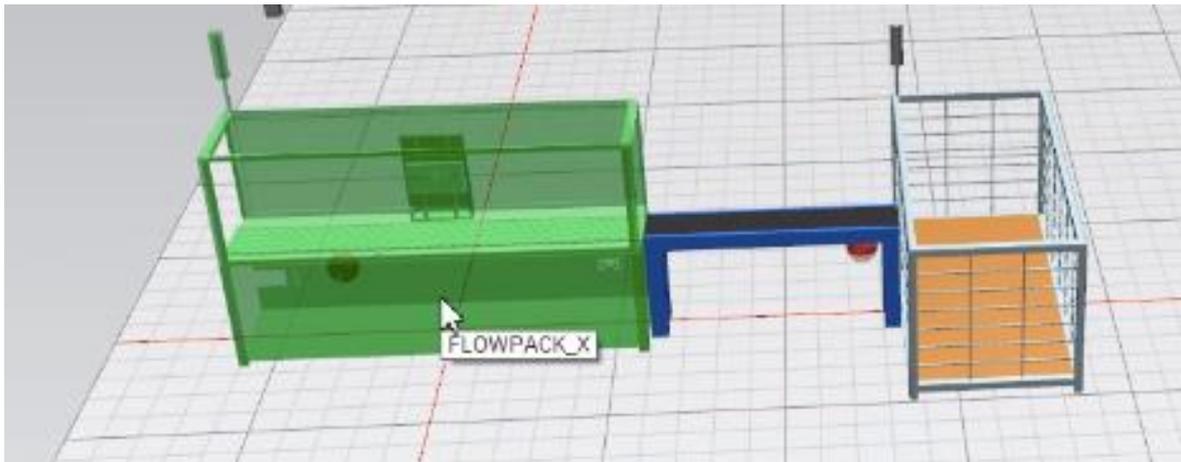


Figure 3.17 FLOWPACK_X station of FLOWPACKX model.

The second block, denoted as F_OUT_X (that stands for FLOWPACK_OUTPUT), represents the conveyor belt on which the final product flows after being flow packed, ready to be collected.

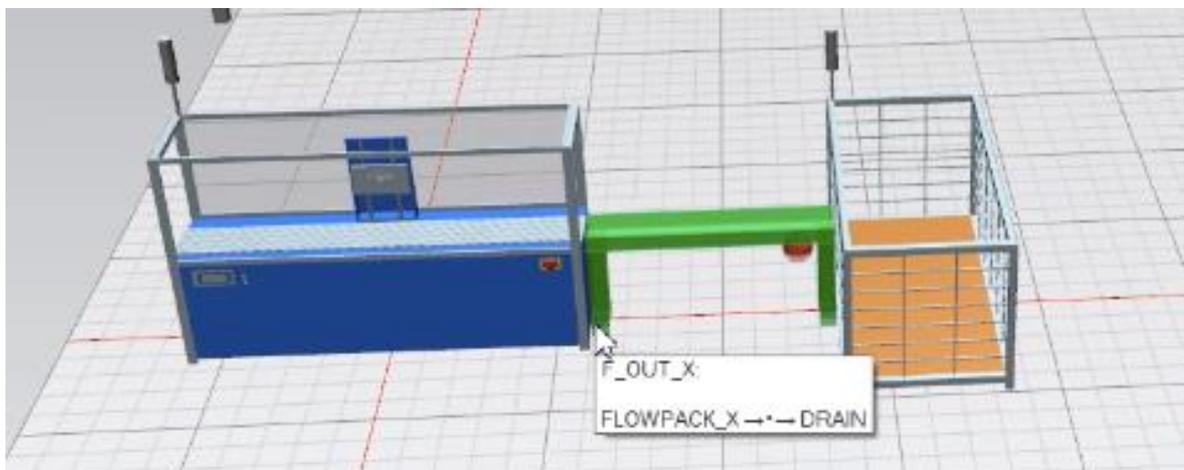


Figure 3.18 F_OUT_X conveyor of LEGX model.

The main attributes of the FLOWPACKX object to be configured are the processing time of the Flowpack and the speed of the F_OUT_X conveyor; again, these parameters are configured according to a specific function called “UpdateSpeed”, as discussed in the subsection about the management of the input parameters.

Apart from that, there are no other relevant methods characterizing the FLOWPACKX class.

3.5 Class “PULLNOSEX”

The PULLNOSEX class represents the object that has the capability of filling the gaps between the different products. According to its real structure, its implementation has been performed exploiting three different conveyor blocks. Anyway, since its real behavior is not representable via software, all the logic has been implemented through a method of the first conveyor block, while the other conveyors do not implement any kind of logic, and so they are not characterized by any methods.

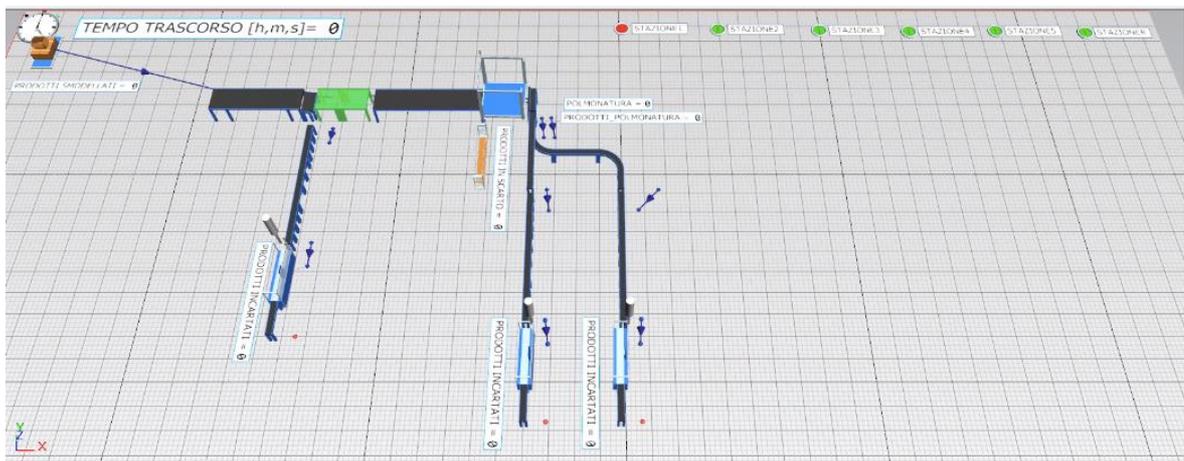


Figure 3.19 PULLNOSEX model in the whole distribution plant.

The first conveyor, denoted as P_P1_X (that stands for PULLNOSE_PARTE1), represents the first belt of the Pullnose and it is characterized by the “OnExit” method.

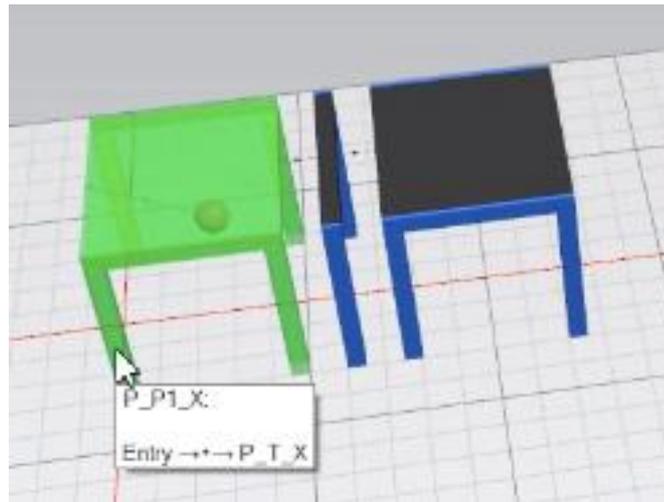


Figure 3.20 P_P1_X conveyor of PULLNOSEX model.

The second conveyor, denoted as P_T_X (that stands for PULLNOSE_TRASLATORE), represents the translator of the Pullnose and it is not characterized by any methods.

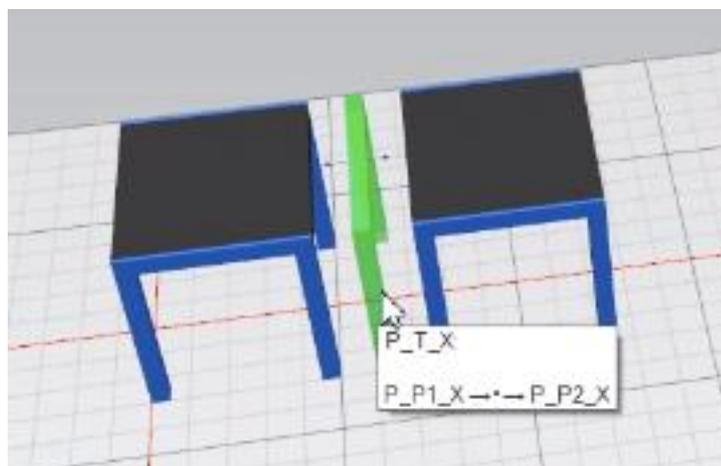


Figure 3.21 P_T_X conveyor of PULLNOSEX model.

The third conveyor, denoted as P_P2_X (that stands for PULLNOSE_PARTE2), represents the second belt of the Pullnose and it is not characterized by any methods.

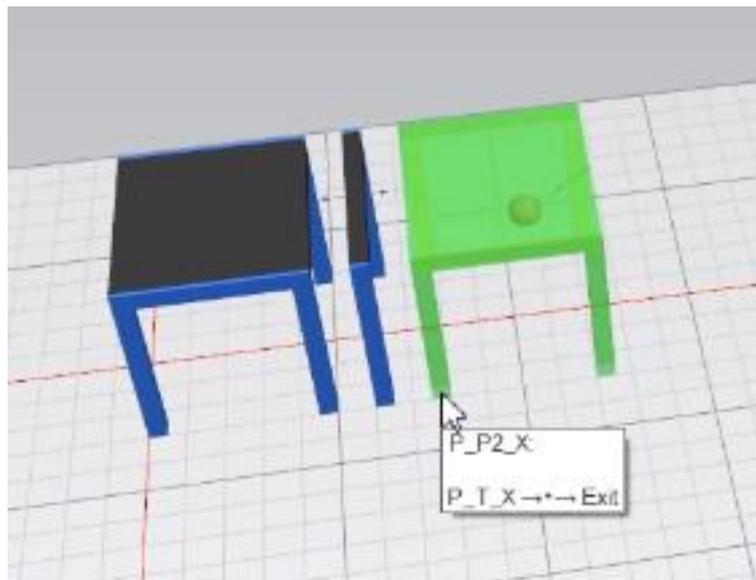


Figure 3.22 P_P2_X conveyor of PULLNOSEX model.

3.5.1 “P_P1” METHODS

Since the real behavior implemented by the translator is not representable via software, a different solution has been adopted, exploiting the graphical functions provided by the platform.

The whole behavior of the PULLNOSEX object has been implemented in the “OnExit” method of the P_P1_X conveyor. The logic is quite simple, and it exploits the functions “deleteobject” and “create”. In particular, performing a for-cycle on the total number of products, they can be deleted one by one while exiting from the P_P1_X conveyor and replaced by new products placed on the P_P2_X conveyor; in this way, exploiting the graphical power of the “create” function, the new products can be placed at specific positions on the P_P2_X conveyor, reproducing the filling of the gaps by which the deleted products were affected while travelling on the P_P1 conveyor.

3.6 Class “INTERASSEBLX”

The INTERASSEBLX class stands for INTERASSE_BUFFER_IN_LINEA and it represents an object that follows the Gondola buffer in the second Plant of interest.

According to its real structure, the INTERASSEBLX object is composed by two floors, each of them characterized by a number of different conveyors.

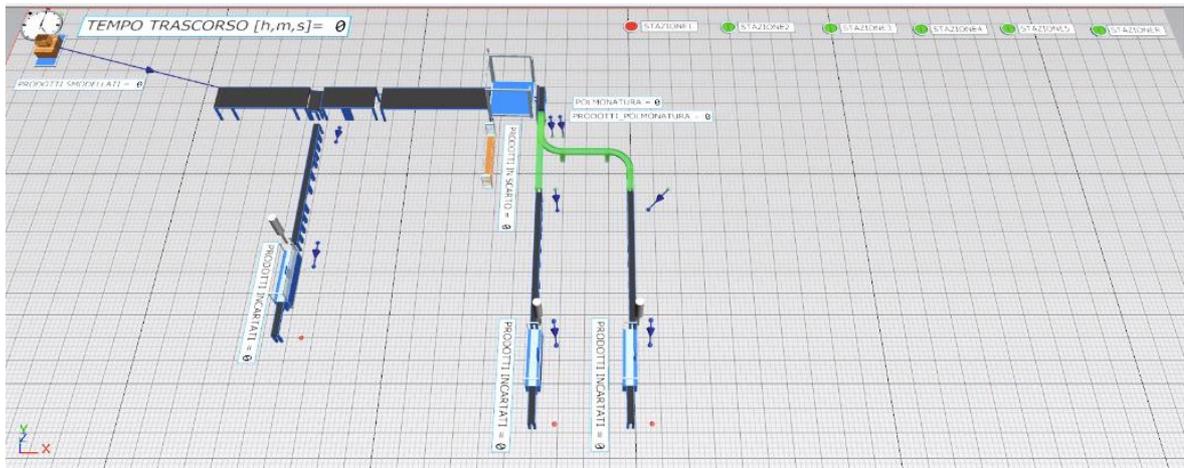


Figure 3.23 INTERASSEBLX model in the whole distribution plant.

The upper floor is composed by one single powered conveyor, called INT_SUP_X.

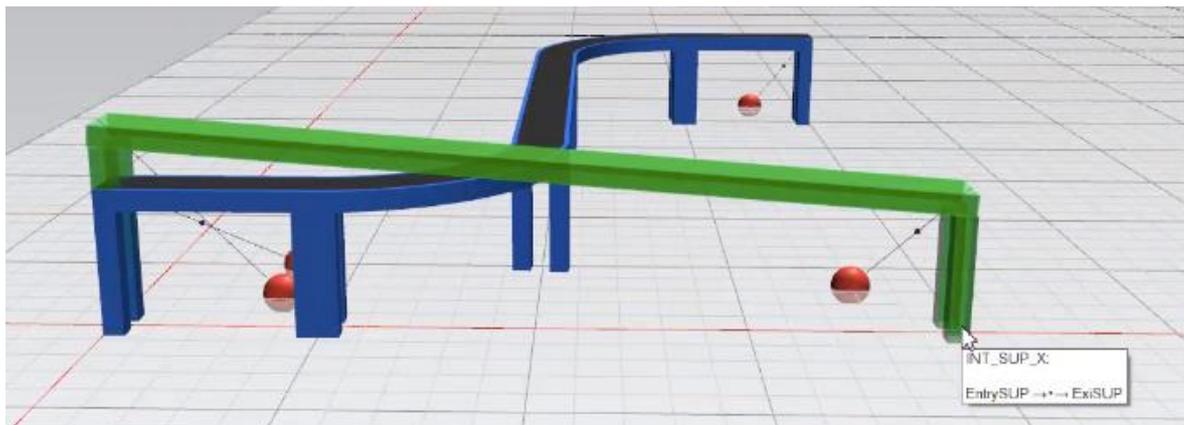


Figure 3.24 INT_SUP_X conveyor of INTERASSEBLX model.

The lower floor is represented as the composition of five different powered conveyors, INT_IN_INF_X, INT_CURVA_SX_INF_X, INT_X, INT_CURVA_DX_INF_X and INT_OUT_INF_X.

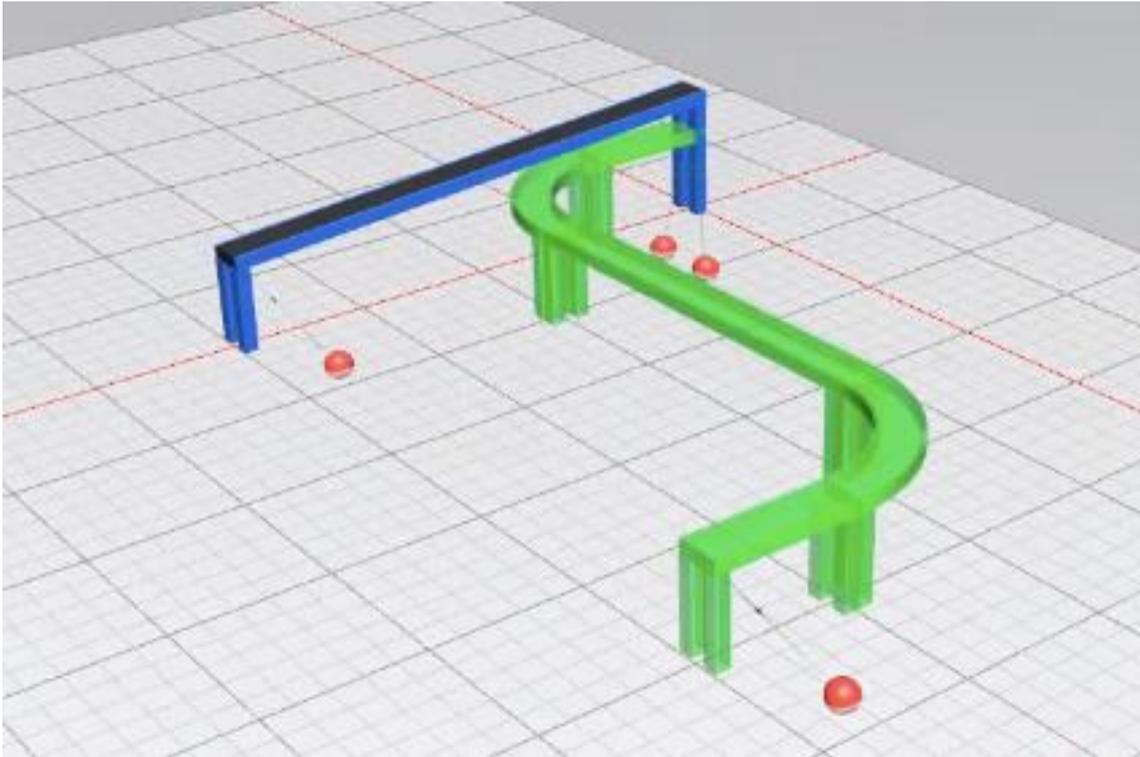


Figure 3.25 INT_IN_INF_X, INT_CURVA_SX_INF_X, INT_X, INT_CURVA_DX_INF_X and INT_OUT_INF_X conveyors of PULLNOSEX model.

The INTERASSEBLX object has been created just to reproduce the real structure of the second distribution plant. In this perspective, it is not characterized by any method and software logic; according to this, the products flow on the different components of the INTERASSEBLX, following the path at the constant nominal speed of 0.4 m/s.

3.7 Class “BUFFERGX”

The BUFFERGX class represents the gondola buffer composing the ridge distribution of the second distribution plant. According to its real structure, its implementation has been performed exploiting five different powered conveyor blocks, one buffer block and one drain block.

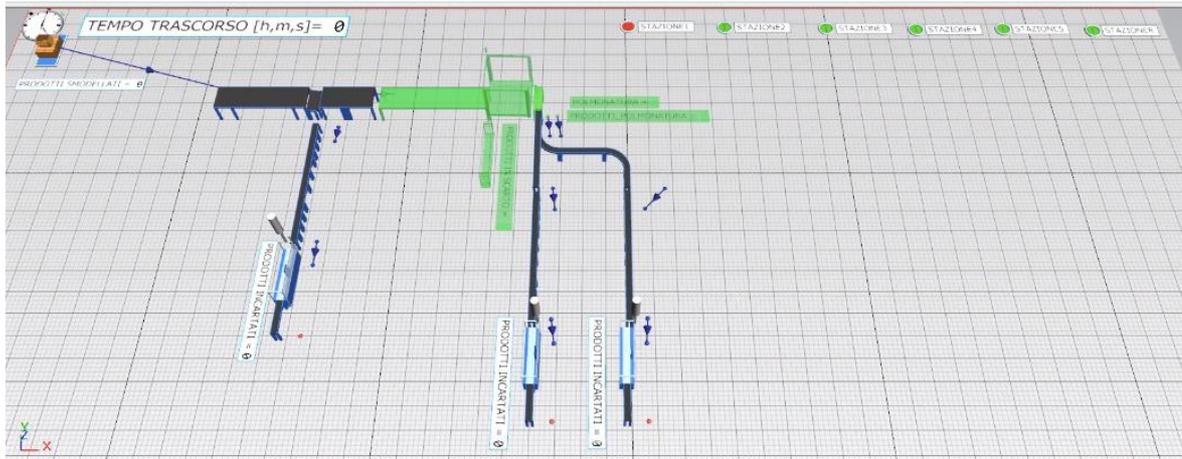


Figure 3.26 BUFFERGX model in the whole distribution plant.

The first block, denoted as D_BUFFER (that stands for DORSALE_BUFFER), represents the conveyor belt on which the product flows before being stored into the actual buffer or, unfortunately, before being discarded. Notice that this component is followed by two successors: one is the actual buffer component, the other one is a drain component.

This block is characterized by the “OnExit” method.

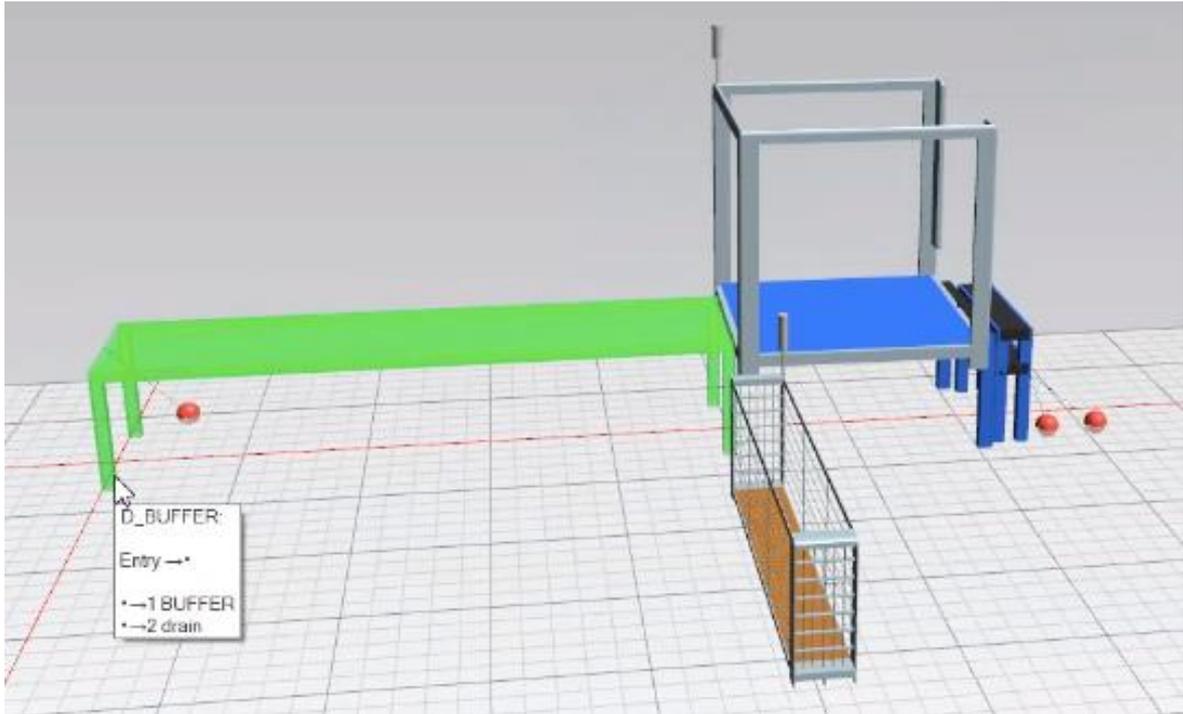


Figure 3.27 D_BUFFER conveyor of BUFFERGX model.

The second block, denoted as BUFFER, belongs to the standard buffer component provided by the software, and it is the one responsible for storing the products and for their distribution.

This block is characterized by the “OnExit” method.

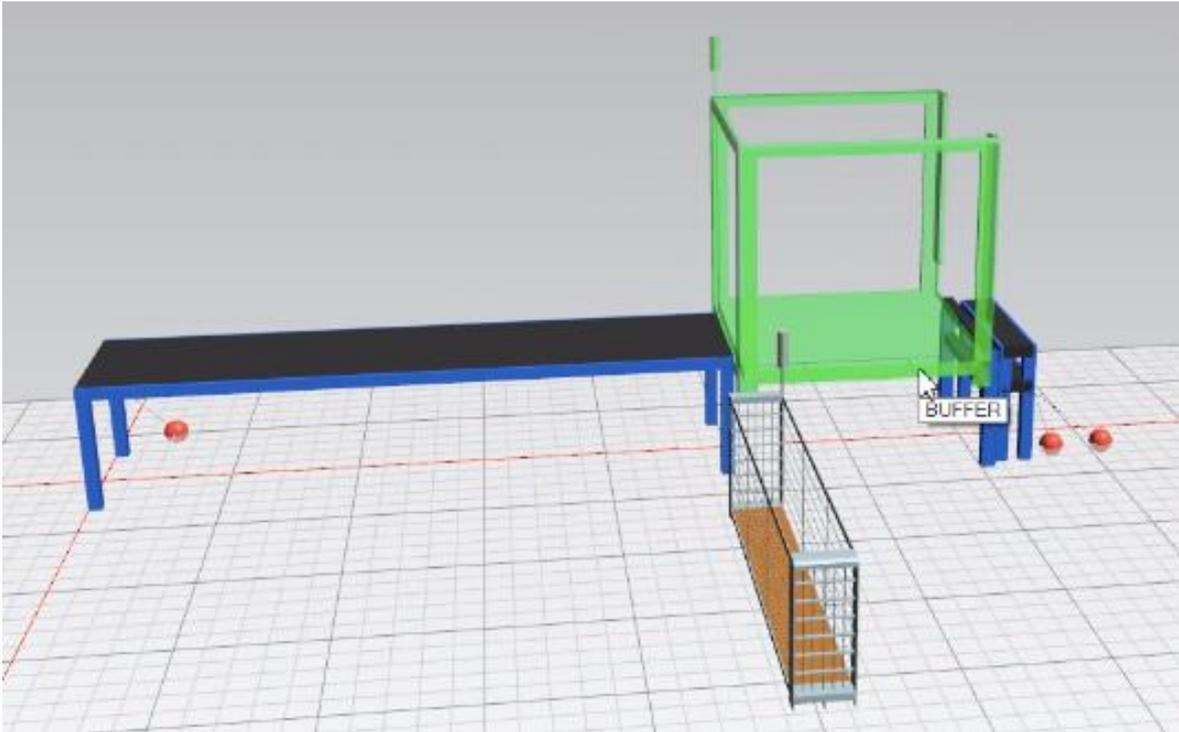


Figure 3.28 BUFFER buffer of BUFFERGX model.

The third and fourth blocks, denoted as `ESTRA_SUP` and `ESTRA_INF` (that stand for `ESTRATTORE_SUPERIORE` and `ESTRATTORE_INFERIORE`), represent the two conveyor belts on which the products flow after being stored.

Both these blocks are characterized by the “OnExit” method.

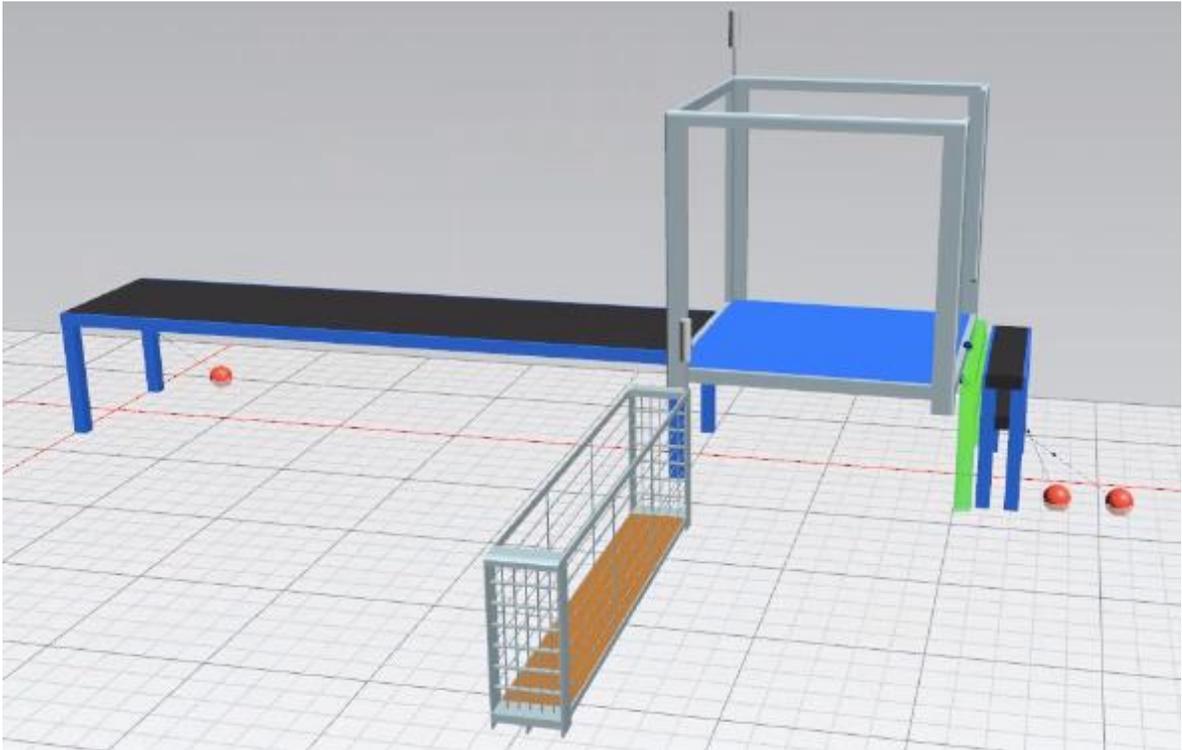


Figure 3.29 ESTRA_SUP and ESTRA_INF conveyors of BUFFERGX model.

The last two conveyors, denoted as CONVEYOR_SUP and CONVEYOR_INF, represent the two conveyor belts that follow the ESTRA_SUP and ESTRA_INF components. They are not characterized by any methods, so the products flow on them at a constant nominal speed.

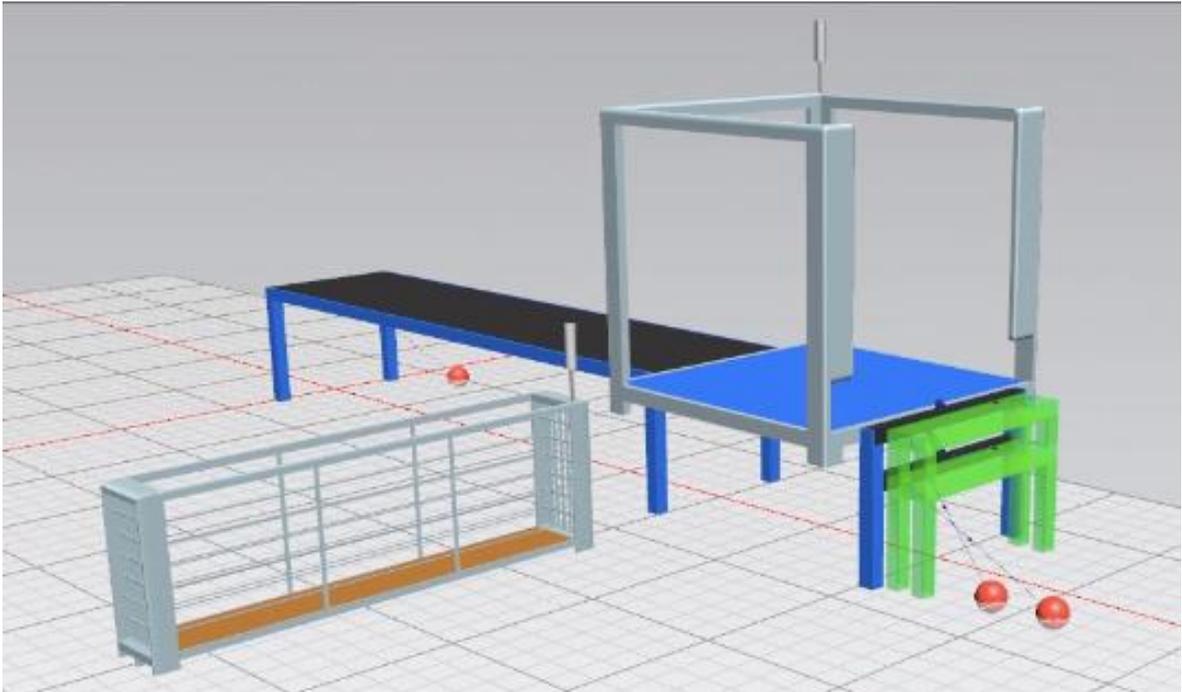


Figure 3.30 CONVEYOR_SUP and CONVEYOR_INF conveyors of BUFFERGX model.

3.7.1 “D_BUFFER” METHODS

The D_BUFFER conveyor is equipped with the “OnExit” method. This method implements the logic through which the products flowing on this conveyor are moved into the actual buffer or discarded flowing into the drain. To do that, a global Boolean variable “scarto” has been defined and initialized to false; it becomes true when the buffer is full and hence it cannot store any products more. Moreover, two additional user-defined attributes of the successive BUFFER component have been exploited; the “nummu” attribute represents the actual number of objects present inside the buffer, the “capacity” attribute represents the maximum number of objects that the buffer is able to store. The value of “scarto” is set through an “if-then-else” logic, according to the following pseudocode:

```

var scarto : boolean
if buffer.nummu > 0.99 * buffer.capacity
    scarto := true
elseif buffer.nummu < 0.99 * buffer.capacity
    scarto := false
end

```

```

if scarto = false
?.mu.move(?.succ(1))
else
?.mu.move(?.succ(2))
end

```

3.7.2 “BUFFER” METHODS

The BUFFER component is equipped with the “OnExit” method. This method implements a controlling function on the operational speed of the Flowpack machines; the logic is based on an “if-elseif” structure, according to the following pseudocode:

```

if ?.nummu > ?.capacity*0.5
root.controls.velocità_flowpack := V_max
elseif ?.nummu < ?.capacity*0.1
root.controls.velocità_flowpack := V_nom
end

```

Notice that the user-defined attribute “velocità_flowpack” of the object FLOWPACK represents the speed in products/min at which the Flowpack works.

Notice that the real mechanism through which the products are alternately moved to the upper and the lower successive conveyors comes for free; indeed, the standard logic of the command “move”, in case of two successors, moves the pieces alternating between the two destinations.

3.7.3 “ESTRA_SUP” AND “ESTRA_INF” METHODS

The methods characterizing the ESTRA_SUP and ESTRA_INF conveyors are specular, in the sense that they are based on the same logic and they perform basically the same actions.

These “OnExit” methods are very similar to the one implemented by the NI_X conveyor of the PIVOTSX object; indeed, also in this case, the unloading of the products is performed in a purely graphical way. According to this graphical approach, first the rank that is going to exit the ESTRA_SUP and ESTRA_INF conveyors is deleted through the function “deleteobject”; then, according to a double for-cycle performed on “RM_NumeroRanghi” and “PR_NumeroProdotti” variables, the previously deleted products are placed on the successive conveyors through the command “create”.

3.8 Class “BUFFERVX”

The BUFFERVX class represents the fan buffer composing the final part of the ridge of the first distribution plant. In this case, two different solutions have been adopted. The first one tries to represent the real structure of the object; then, because of a lack of time, this solution is still incomplete in the recycling behavior and so it can’t be used in the implementation of the whole distribution plant. For this reason, a second solution has been implemented; this solution is simplified, in the sense that it focuses on the functionality of the buffer without taking care of the real structure of the object.

In the following, the first solution will be briefly sketched, while the second one will be fully illustrated and discussed.

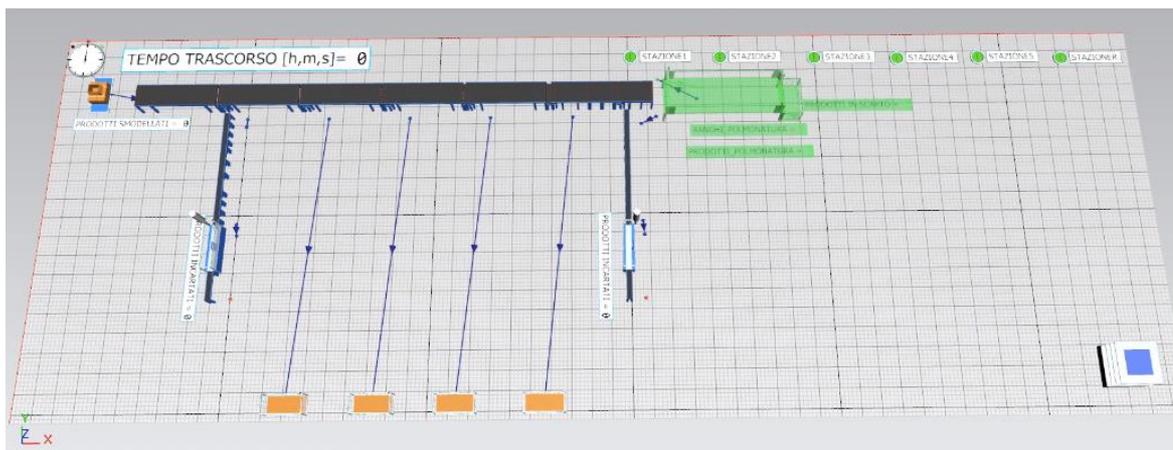


Figure 3.31 BUFFERVX model in the whole distribution plant.

- **FIRST SOLUTION**

The first solution aimed at representing the fan buffer according to its real structure; to do that, its implementation has been performed exploiting a number of powered conveyor blocks. In particular, the buffer is composed by a first conveyor followed by a number of different conveyors representing the floors of the fan buffer; the number of the floors is parametrizable by the user, in the figure 3.32 it is assumed equal to four.

The first conveyor represents the final segment of the ridge distribution preceding the different floors. For what concerns the nominal direction, this conveyor has the capability of pivoting in order to unload the products on the different floors; for what concerns the backwards direction, it must be able to perform the recycling according to a certain logic. To do these operations, the conveyor must be equipped with two sensors, besides the “OnEntrance”, “OnExit” and “OnBackwardExit” methods.

Also the floors are represented by powered conveyors, equipped with a sensor, besides the “OnEntrance” and “OnBackwardExit” methods. The main functionality that the floors have to implement are the step pace for the nominal direction and the recycling for the backwards direction.

As previously announced, due to a malfunctioning in the recycling behavior of the object, the company decided to temporarily abandon this implementation, focusing on a simpler object able to guarantee the simulation of the whole distribution plant and so providing some useful results in terms of performance analysis. In the following, this simplified implementation will be presented and discussed.

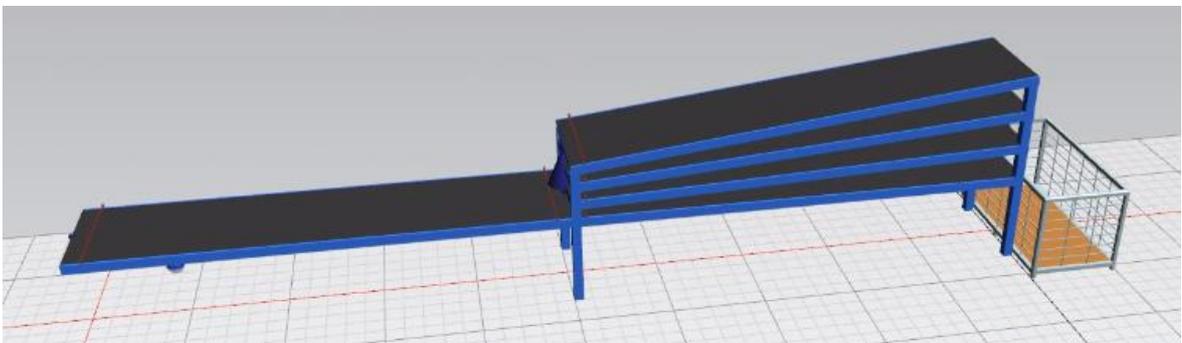


Figure 3.32 First solution of BUFFERVX model.

- ***SECOND SOLUTION***

The second solution focuses on the functionalities of the fan buffer, without taking care of its real structure. In this perspective, its implementation has been performed exploiting two powered conveyor blocks, one buffer block and one drain block.

The first block, denoted as BUFFER, belongs to the standard buffer component provided by the software and it is the one responsible for storing the products and for their distribution.

This block is characterized by the “OnObserver” method.

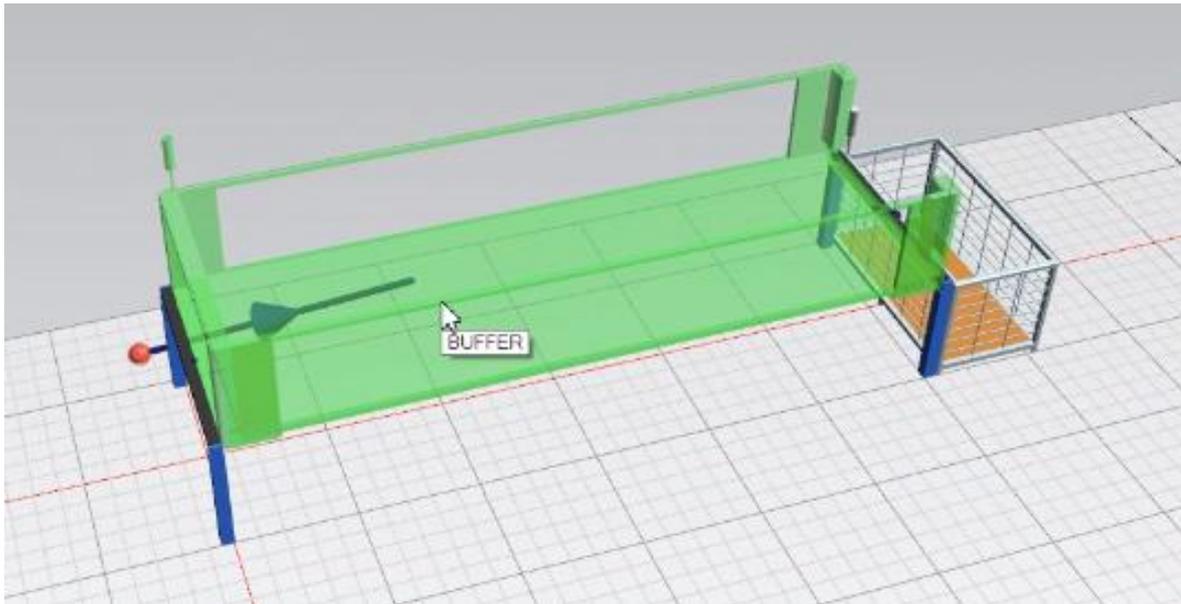


Figure 3.33 BUFFER buffer of BUFFERVX model.

The first conveyor, denoted as USCITABUFFER_X, represents the path that the product follows when it is going to be recycled.

This block is characterized by the “OnExit” method.

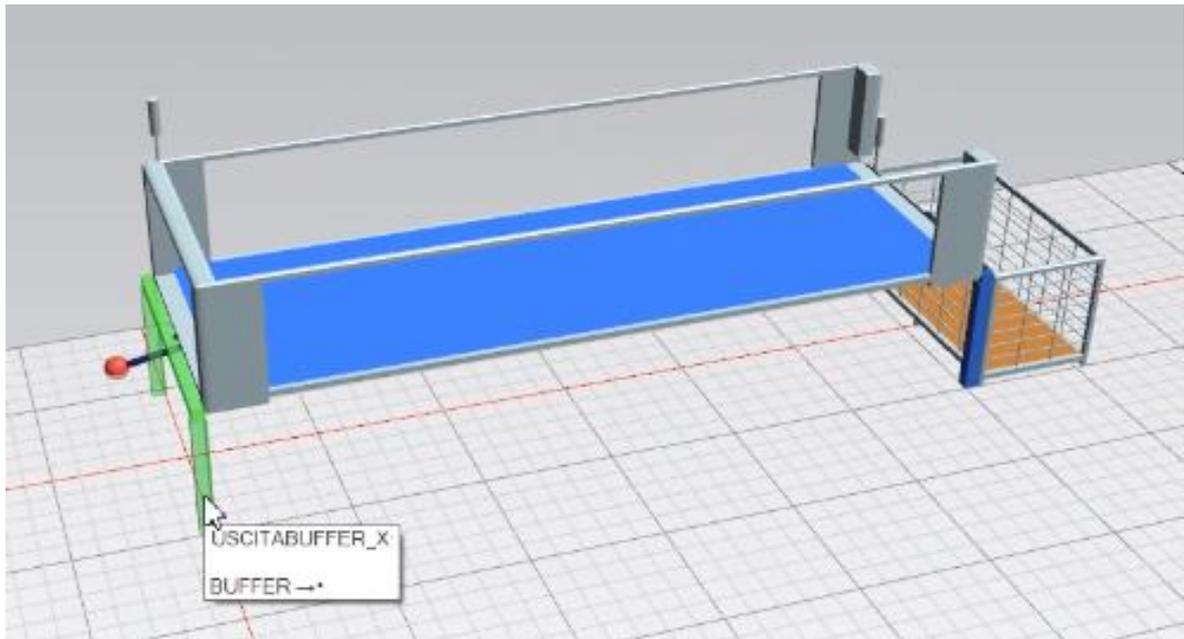


Figure 3.34 USCITABUFFER_X conveyor of BUFFERVX model.

The second conveyor, denoted as USCITASCARTO_X, represents the path that the product follows when it is going to be discarded.

This block is not characterized by any methods, so the products flow on it at the nominal speed towards the drain.

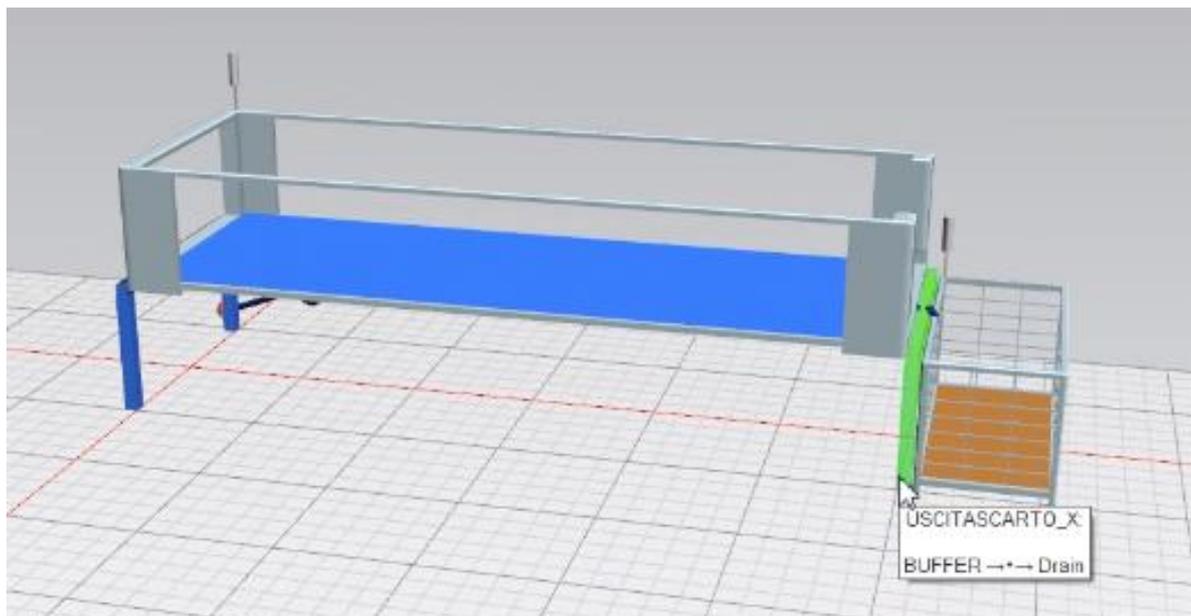


Figure 3.35 USCITASCARTO_X conveyor of BUFFERVX model.

3.8.1 “BUFFER” METHODS

The BUFFER component is equipped with an “OnObserver” method. This method is called whenever the value of the attribute “nummu” of BUFFER changes and it performs the moving of the products according to a simple if-elseif logic.

In particular, if “nummu” is greater than or equal to the capacity of the buffer, then the Boolean attribute “entrancelocked” of USCITASCARTO_X is set to true; in this way, until it is not full, the buffer has only the possibility of moving the pieces to USCITABUFFER, performing the recycle. Then, elseif “nummu” is greater than the capacity of the buffer, the Boolean attribute “entrancelocked” of USCITASCARTO_X is set to false, the attribute “Speed” of USCITASCARTO_X is set to the nominal speed and the product is moved to that conveyor through the command “move”, ready to be discarded towards the drain.

3.8.2 “USCITABUFFER_X” METHODS

The USCITABUFFER_X component is equipped with the “OnExit” method and it implements the recycle mechanism moving the product on the PIVOTDX. To do that, it waits until the number of products, represented by the attribute “nummu” of the conveyor D_CR of the PIVOTDX object, is less than or equal to 3 and then it moves the product on it through the command “insert”. The value 3 represents the maximum capacity that allows the DDD conveyor of PIVOTDX to have the space for receiving the product according to its length and its speed.

Chapter 4 - Input and output management

In both the considered plants, a frame called “controls” is inserted as instance. This frame is built in the class library, in the same modality it has been adopted for all the other objects.

This frame contains all the methods and the tables and variables that are used for the input and output data management. In particular, in figure 4.1 the instance of this frame and its content are shown, where the instance of the frame is indicated by a red arrow:

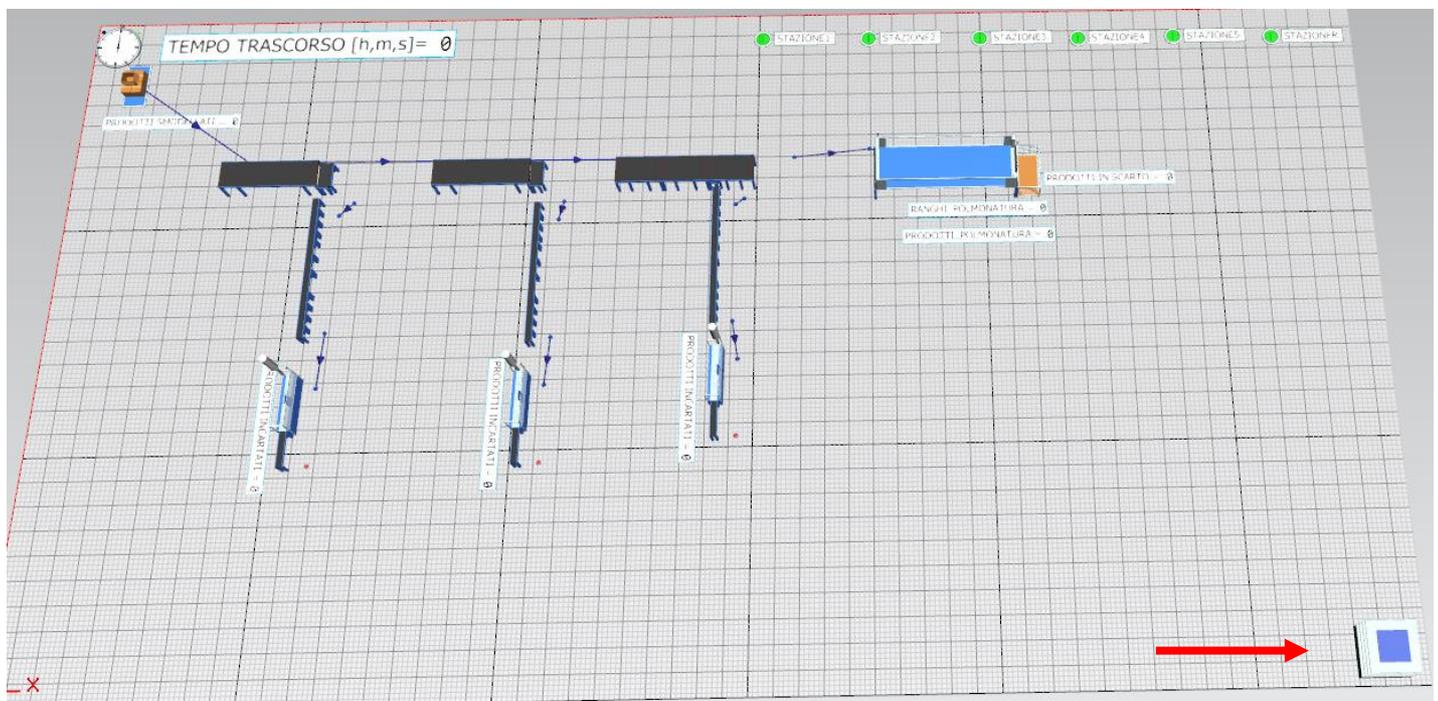


Figure 4.1 Instance of the frame "controls"

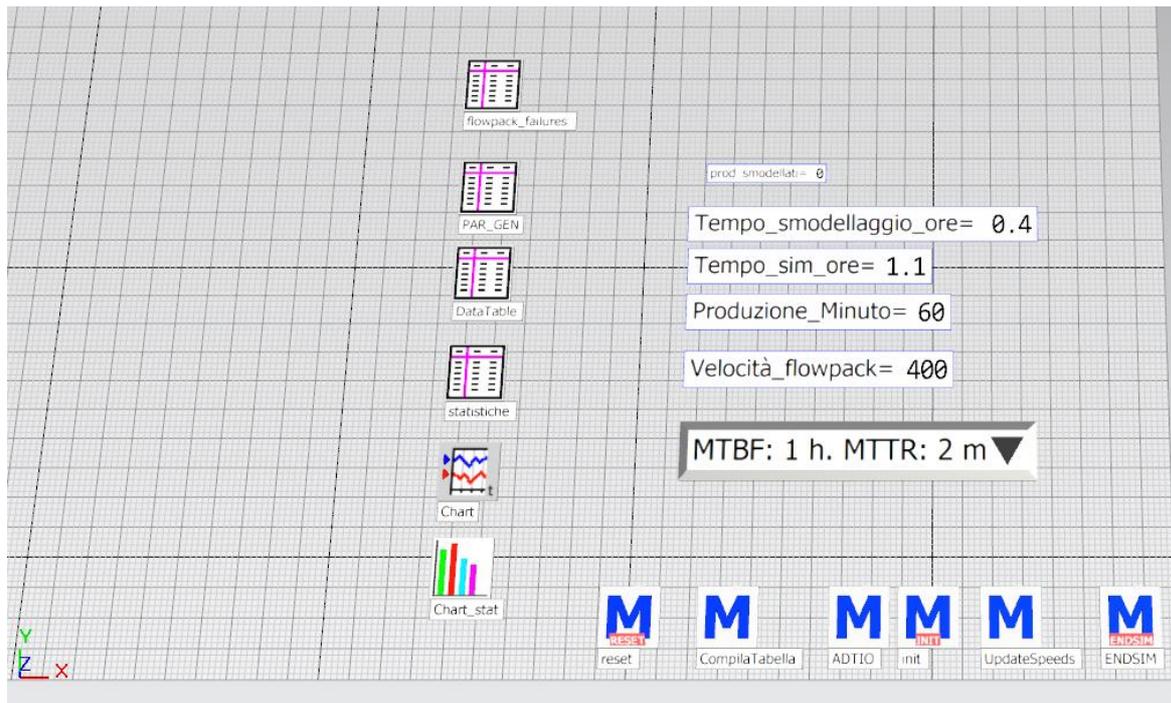


Figure 4.2 Content of the frame controls

4.1 Input management

First of all, it can be seen that in the principal frame, which is called root frame, there is a display showing the so called “TEMPO TRASCORSO”, which is a display that keeps track of the simulation time and six buttons located on the upper right corner. These buttons can switch to red or green depending on the opening or closure of the charts related to each single and double pivot. In particular, the green colour means that the chart is able to receive products and deliver them to the following leg; the red colour means that the chart cannot receive any product and so the flowpack of that portion of the line, depending on that chart, cannot wrap any product.

4.1.1 FRAME CONTROLS CONTENT

In this frame many methods, tables, charts and variables are contained that are used for every plant distribution considered. The way to make it functioning is to create an instance of this frame in the considered root frame and then introduce any desired modification to the tables and variables contained in it. By taking into account only the elements belonging to this frame and related to the inputs, the following classification can be performed:

- The table **PAR_GEN**.
- The table **DataTable**.
- The method “**Update speeds**”.
- The variables **Tempo_smodellaggio_ore**, **Tempo_sim_ore** and **Produzione_minuto**.

The table **PAR_GEN** contains only the information related to the products, i.e, how many products compose a rank, how many ranks compose a mould, the products length, width and height and so on.

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
1	RISERVA_1							prodotto			
2	RISERVA_2							prodotto			
3	RISERVA_3							prodotto			
4	L_LunghezzaProdotti	90						prodotto			
5	W_LarghezzaProdotti	30						prodotto			
6	H_AltezzaProdotti	11.7						prodotto			
7	P_DistanzaProdotti	2.5						prodotto			
8	R_DistanzaRanghi	5						prodotto			
9	M_DistanzaStampi	0						prodotto			
10	PR_NumeroProdotti					20		prodotto			
11	RM_NumeroRanghi					1		prodotto			
12	RISERVA_12							prodotto			
13	RISERVA_13							prodotto			
14	RISERVA_14							prodotto			
15	RISERVA_15							prodotto			
16	RISERVA_16							prodotto			
17	MUWidth	647.5						prodotto			
18	MULength	90						prodotto			
19	MUHeight	11.7						prodotto			
20	RISERVA_20							prodotto			
21	RISERVA_21							prodotto			
22	RISERVA_22							prodotto			
23	RISERVA_23							prodotto			

Table 4.1 Table PAR_GEN content

Table 4.1, as all the others built in the software, contains 64 variables of interest, but only some of them have been initialized and used. All the others labelled with the name “RISERVA_NUM” can be initialized in the future for further developments.

The link to this table is inserted in the source of each considered root frame; in this way, the products coming out from the source itself has all the attributes defined in the PAR_GEN table as user defined attribute.

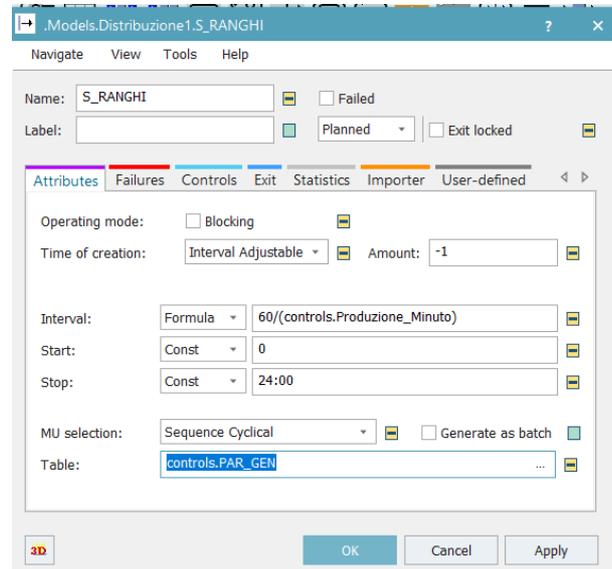


Figure 4.3 Path to PAR_GEN in the source

Furthermore, another display is located immediately below the source, which is identified as “PRODOTTI SMOCELLATI” and it indicates the number of products which are emitted by the source in real time.

The data defined in the table PAR_GEN are then recalled in the “reset” method, in order to automatically assign certain dimensions to the mobile units coming from the source when the simulation starts, taking into account the gaps, the number of products per rank and the number of ranks per mould. This is done by considering the following formulas and variables:

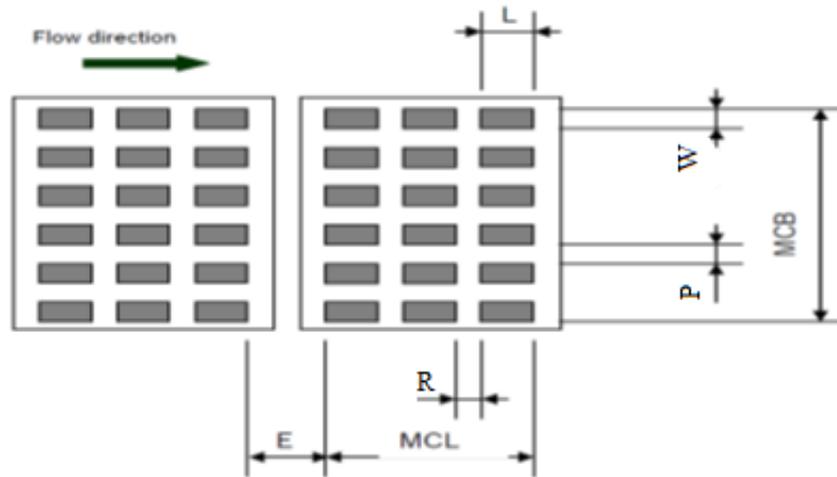


Figure 4.4 Variables of interest

$$MobileUnitWidth = PR * W + (PR - 1) * P \quad (4.1)$$

$$MobileUnitLength = RM * L + (RM - 1) * R \quad (4.2)$$

Where:

- **PR:** products per row;
- **RM:** rows per mould;
- **P:** distance between products in a row;
- **R:** distance between rows;
- **W:** product width;
- **L:** products length.

In table 4.2 it is shown how the data inserted in the PAR_GEN table automatically determine a change in the mobile unit coming out from the source:

string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String
1	RISERVA_1							prodotto
2	RISERVA_2							prodotto
3	RISERVA_3							prodotto
4	L_LunghezzaProdotti	90						prodotto
5	W_LarghezzaProdotti	30						prodotto
6	H_AltezzaProdotti	11.7						prodotto
7	P_DistanzaProdotti	2.5						prodotto
8	R_DistanzaRanghi	5						prodotto
9	M_DistanzaStampi	0						prodotto
10	PR_NumeroProdotti					20		prodotto
11	RM_NumeroRanghi					1		prodotto
12	RISERVA_12							prodotto
13	RISERVA_13							prodotto
14	RISERVA_14							prodotto
15	RISERVA_15							prodotto
16	RISERVA_16							prodotto
17	MUWidth	647.5						prodotto
18	MULength	90						prodotto
19	MUHeight	11.7						prodotto
20	RISERVA_20							prodotto
21	RISERVA_21							prodotto
22	RISERVA_22							prodotto
23	RISERVA_23							prodotto

Table 4.2 Mobile unit as a rank

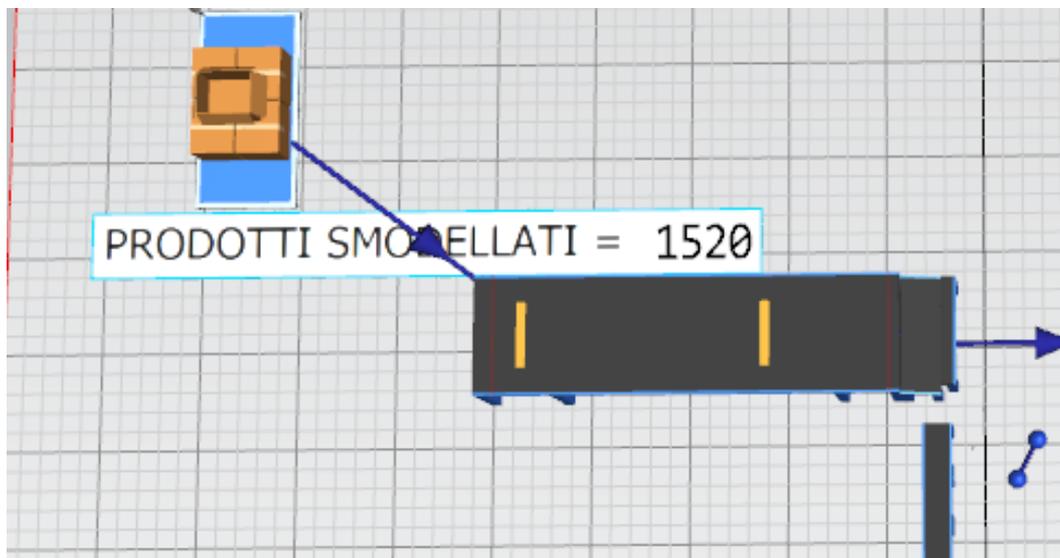


Figure 4.5 Rank exiting from the source

1	2	3	4	5	6	7	8	
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String
1	RISERVA_1							prodotto
2	RISERVA_2							prodotto
3	RISERVA_3							prodotto
4	L_LunghezzaProdotti	200						prodotto
5	W_LarghezzaProdotti	30						prodotto
6	H_AltezzaProdotti	11.7						prodotto
7	P_DistanzaProdotti	2.5						prodotto
8	R_DistanzaRanghi	5						prodotto
9	M_DistanzaStampi	0						prodotto
10	PR_NumeroProdotti					20		prodotto
11	RM_NumeroRanghi					2		prodotto
12	RISERVA_12							prodotto
13	RISERVA_13							prodotto
14	RISERVA_14							prodotto
15	RISERVA_15							prodotto
16	RISERVA_16							prodotto
17	MUWidth	647.5						prodotto
18	MULength	405						prodotto
19	MUHeight	11.7						prodotto
20	RISERVA_20							prodotto
21	RISERVA_21							prodotto
22	RISERVA_22							prodotto
23	RISERVA_23							prodotto

Table 4.3 Mobile unit as a mould

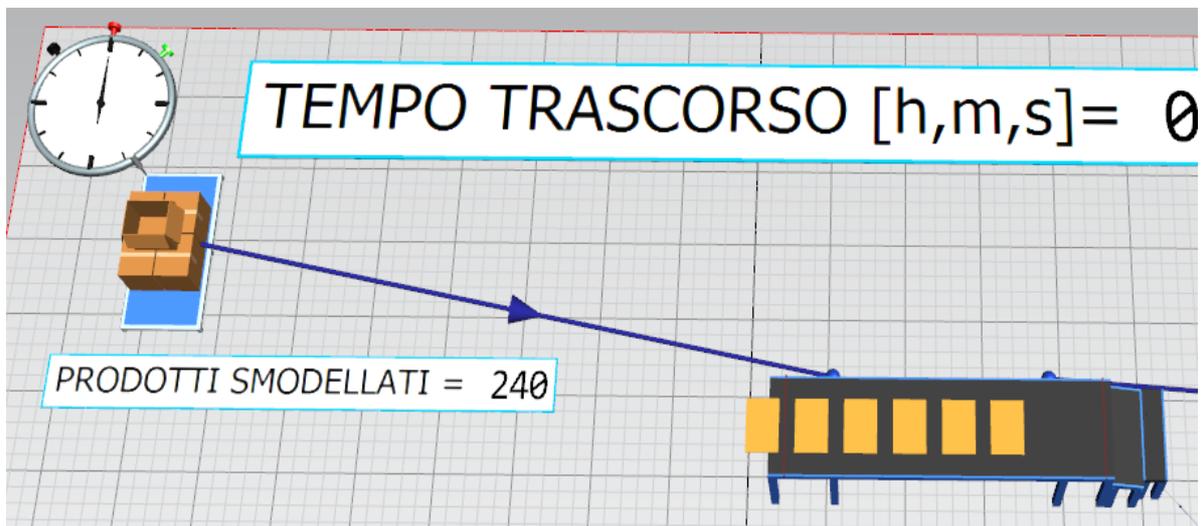


Figure 4.6 Mould exiting from the source

In the table **DataTable** all the instances of the objects inserted in the root frame are automatically written time by time. By considering the frame with the simple buffer in it, the DataTable is constructed as in table 4.4.

	string 1	object 2	table 3	5 4
string	nome oggetto	oggetto	attributi	
1	FLOWPACK1	*.Models.Distribuzione100.FLOWPACK1	Lista Attributi	
2	FLOWPACK6	*.Models.Distribuzione100.FLOWPACK6	Lista Attributi	
3	PIVOTS2	*.Models.Distribuzione100.PIVOTS2	Lista Attributi	
4	PIVOTS3	*.Models.Distribuzione100.PIVOTS3	Lista Attributi	
5	PIVOTS4	*.Models.Distribuzione100.PIVOTS4	Lista Attributi	
6	PIVOTSS	*.Models.Distribuzione100.PIVOTSS	Lista Attributi	
7	PIVOTDR	*.Models.Distribuzione100.PIVOTDR	Lista Attributi	
8	PIVOTS1	*.Models.Distribuzione100.PIVOTS1	Lista Attributi	
9	BUFFERVSEMPlice	*.Models.Distribuzione100.BUFFERVSEMP...	Lista Attributi	
10	LEG1	*.Models.Distribuzione100.LEG1	Lista Attributi	
11	LEG6	*.Models.Distribuzione100.LEG6	Lista Attributi	
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				

Table 4.4 DataTable content

This table is automatically built by means of the method called “Compila tabella”, which allows to create exactly the format of the table shown in Table 4.4: in particular, it writes in the first column the name of the instance present in the root frame; in the second column, the object itself is written, characterized by its path; finally, in the third column, a nested list of the attributes associated to the instance is created. This nested list can be shown by double clicking on the name “Lista Attributi”, which opens a new window in which all the attributes given to the single class of the considered object appear.

These attributes are the ones defined in the table “attributi” present in each class of objects, so they are user-defined. An example related to the PIVOTS is showed in table 4.5.

	string	length[mm]	speed[m/min]	acceleratio	real	integer	boolean	string	time
1	2	3	4	5	6	7	8	9	
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time
1	Lunghezza_nastro_1	2000						PIVOTS	
2	Larghezza_nastro_1	1200						PIVOTS	
3	velocita_avanti_nastro_1		24					PIVOTS	
4	velocita_indietro_nastro_1		24					PIVOTS	
5	RISERVA_05							PIVOTS	
6	RISERVA_06							PIVOTS	
7	RISERVA_07							PIVOTS	
8	RISERVA_08							PIVOTS	
9	Lunghezza_nastro_pivot	4200						PIVOTS	
10	Larghezza_nastro_pivot	1200						PIVOTS	
11	velocita_avanti_nastro_pivot		24					PIVOTS	
12	velocita_indietro_nastro_p...		24					PIVOTS	
13	RISERVA_13							PIVOTS	
14	RISERVA_14							PIVOTS	
15	RISERVA_15							PIVOTS	
16	RISERVA_16							PIVOTS	
17	Lunghezza_nastro_pivot_ric	2000						PIVOTS	
18	Larghezza_nastro_pivot_ric	1200						PIVOTS	
19	velocita_av_nastro_pivot_ric		24					PIVOTS	
20	velocita_in_nastro_pivot_ric		24					PIVOTS	
21	RISERVA_21							PIVOTS	
22	RISERVA_22							PIVOTS	
23	RISERVA_23							PIVOTS	

Table 4.5 Part of the attributes of the instance PIVOTS1

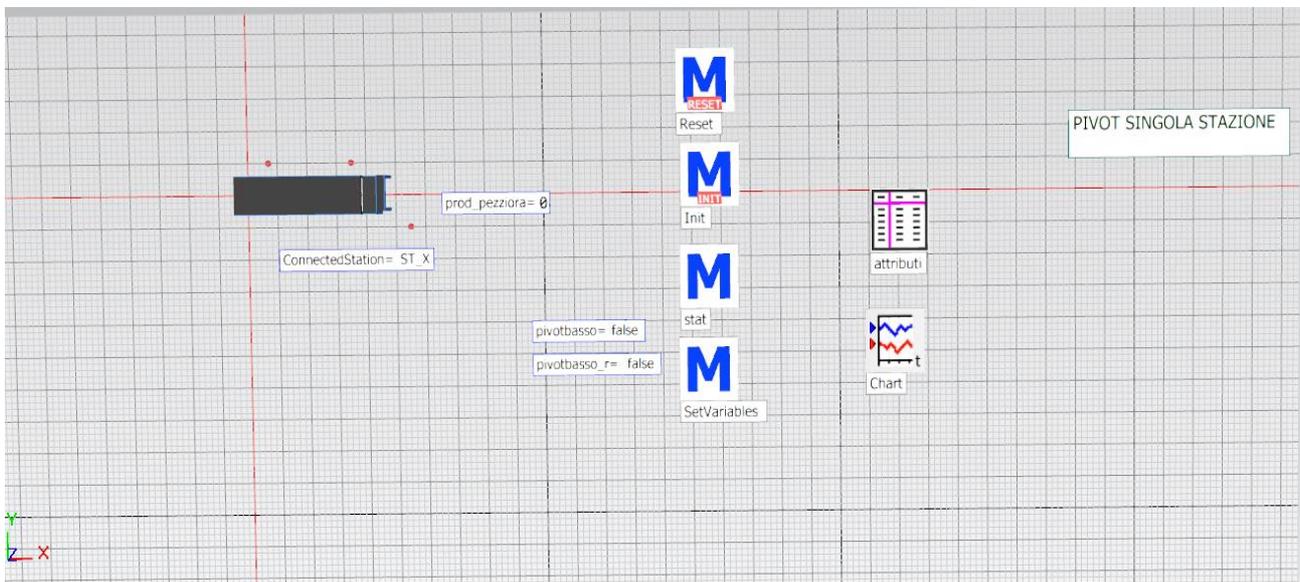


Figure 4.7 Table "attributi" present in the class PIVOTSX

Once the table DataTable has been compiled, thanks to its structure, it is possible to define new attributes with a certain value assigned to them, which are only defined for a specific instance of the class. In this way, the original class is not affected by the apported modification, as well as all the other instances of the same class of objects.

This is done in order to have the possibility to differentiate the various instances of the same object class inserted in the frame, in order to be able to assign them different characteristics, by enriching in this way the table “attributi” belonging to the instance subject to the apported modifications performed by the DataTable modification.

These modifications are introduced by means of the “ADTIO” method, which is run every time the table DataTable is closed after the modifications have been apported, as it can be seen by the linking between the name of the method and the location in which it is recalled in Figure 4.8.

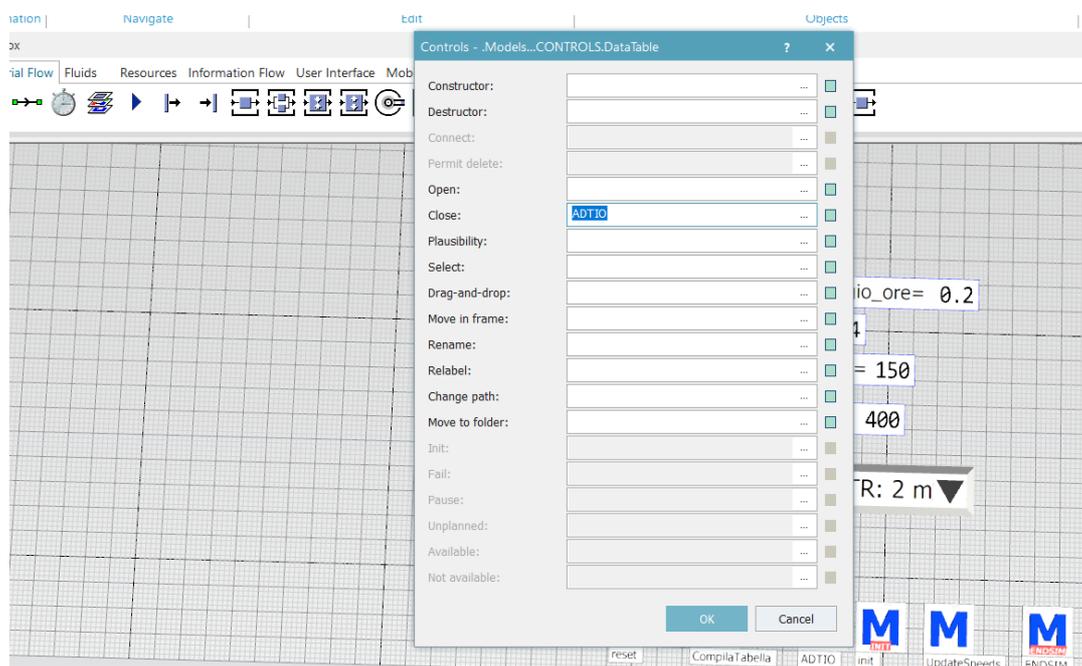


Figure 4.8 Linking between the closing event of DataTable and the method ADTIO

The “**Update speeds**” is a method that has the specific task to call every other existing method called “Update speed” existing in the different objects inserted in the frame, in order to have the update of the speeds of the various elements of the frame that need a change in their speed, such as the legs and the flowpacks. Along with this method, the variable “Velocità_flowpack” is defined, which is related to the speed of the flowpack and it is used in the methods that are present in the flowpack object in order to suitably assign a velocity to the flowpack itself.

The variables **Tempo_smodellaggio_ore**, **Tempo_sim_ore** and **Produzione_minuto** are related to the time to stop the source in the products emission (expressed in hours), the time to stop the simulation (expressed in hours) and the pieces (intended as mobile units) emitted minute by minute by the source, respectively. In particular, the variables **Tempo_smodellaggio_ore** and **Tempo_sim_ore** have been created in order to let the simulation stop after that the source has stopped producing mobile units, allowing the plant

to absorb all the remaining pieces laying on the line, while the variable `Produzione_minuto` is used in order to correctly feed all the legs and flowpacks.

4.1.2 INSIGHT IN THE OBJECTS CLASSES: RESET METHOD

All the classes of the different objects have been provided with a predefined set of methods and tables and enriched with other methods specifically designed for the considered object.

Every class has the methods `Reset` and `Init` and, obviously, the table `attributi`.

By taking into consideration the frame of the double pivot `PIVOTDX`, the resulting structure is the one recalled in figure 4.9.

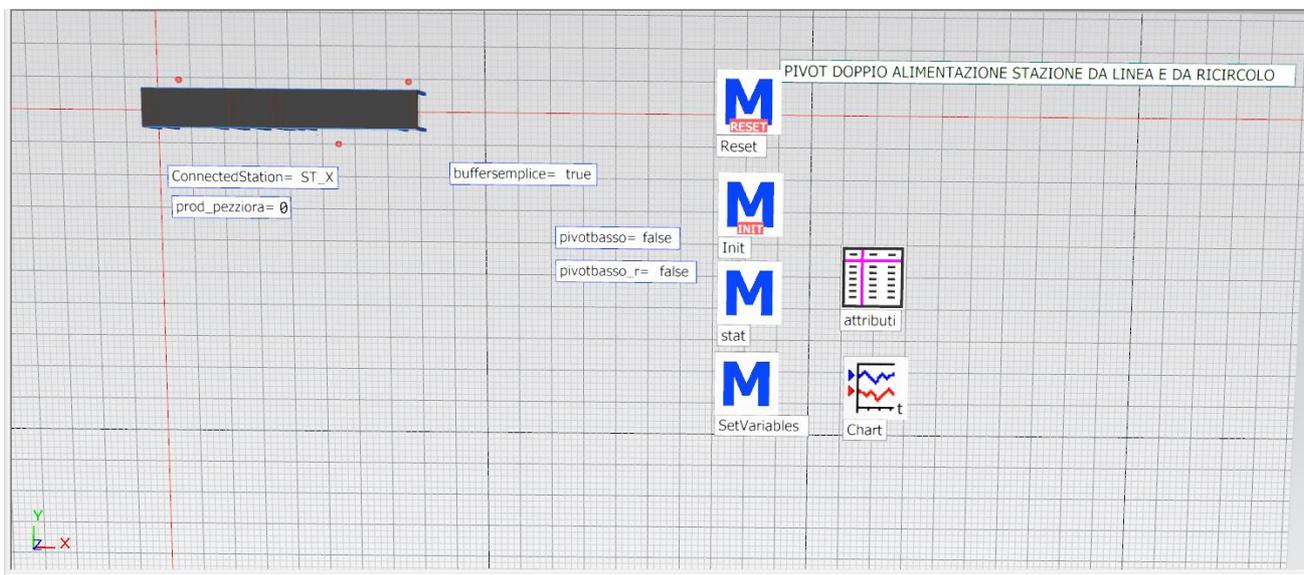


Figure 4.9 Structure of the frame PIVOTDX

In the **Reset** method of each object the following operations are performed:

- **Dimensioning of the different conveyors** by pointing to the right cells of the table `attributi`: in particular, the belts considered in the pivots, both `PIVOTSX` and `PIVOTDX` are made up of three different segments for each belt. In the reset code only the central segment is modified by taking into account the length value indicated in the table `attributi` and then also the width is correctly pointed in the specific cell of the same table. In this way, the conveyors are automatically dimensioned only by putting values in the table `DataTable` in the frame `Controls` and, when the closing of this table happens, the value automatically appears in the table attribute of the considered instance and with the reset method assume the dimensions indicated in the table are automatically assigned to the conveyor. In figure 4.10, this modification apported to the firs PIVOTS are shown in sequence.

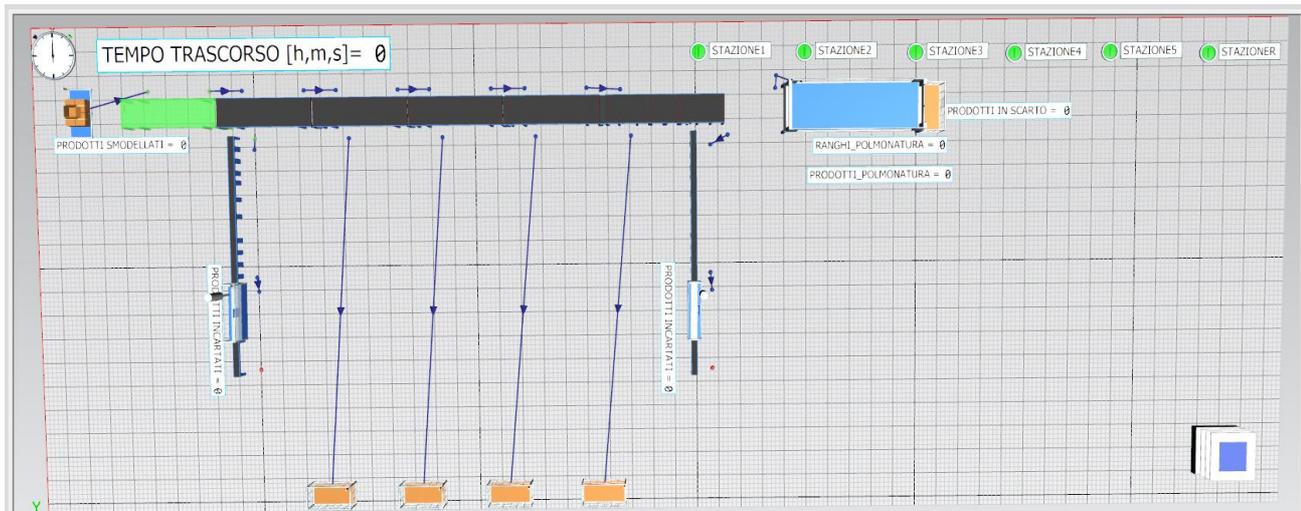


Figure 4.10 Plant with PIVOTS1 first belt dimensions not modified (value of length in the table: 4200 mm)

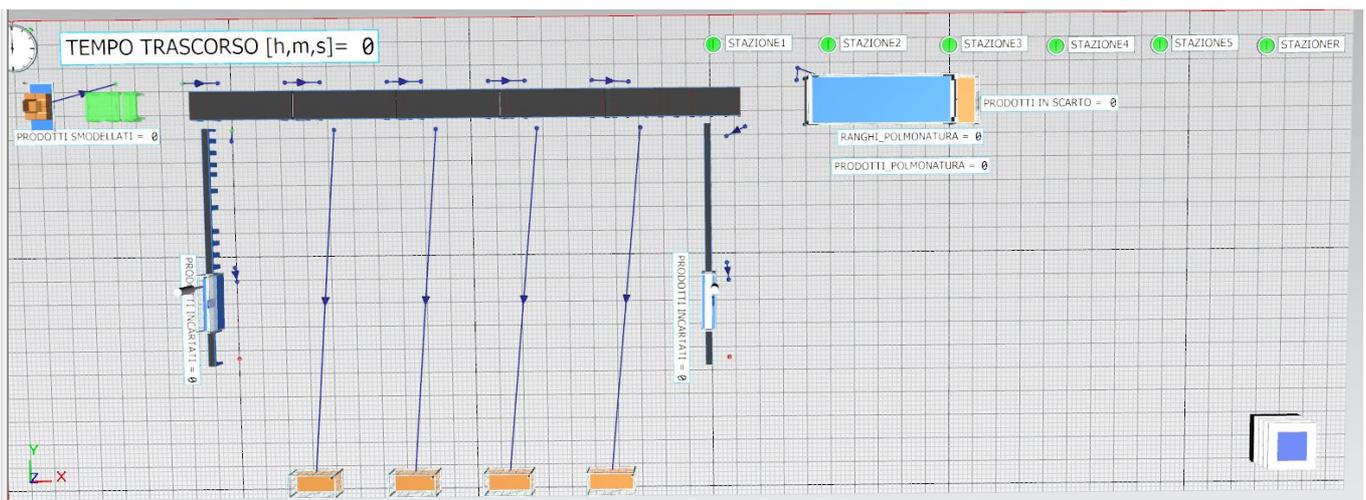


Figure 4.11 PIVOTS1 first belt length after modification in DataTable (new length: 200 mm)

- Definition of the origin position:** when the conveyors change dimensions, all the following conveyors belonging to the same object must be suitably shifted forward or backward in order to keep the right distances among the different elements that compose the global object. In order to reach this goal, a code that regulates the starting point of each conveyor is implemented. The resulting effect is visible in figure 4.11, where even if the length of one belt diminishes, the following conveyors are suitably moved by keeping all the elements compact.

- **Definition of the left or right exit:** the last portion of the reset is dedicated to a suitable turning of the different elements of the plant, if the moving direction of the product is on the right side or on the left side. Figures 4.12 and 4.13 represent such a situation.

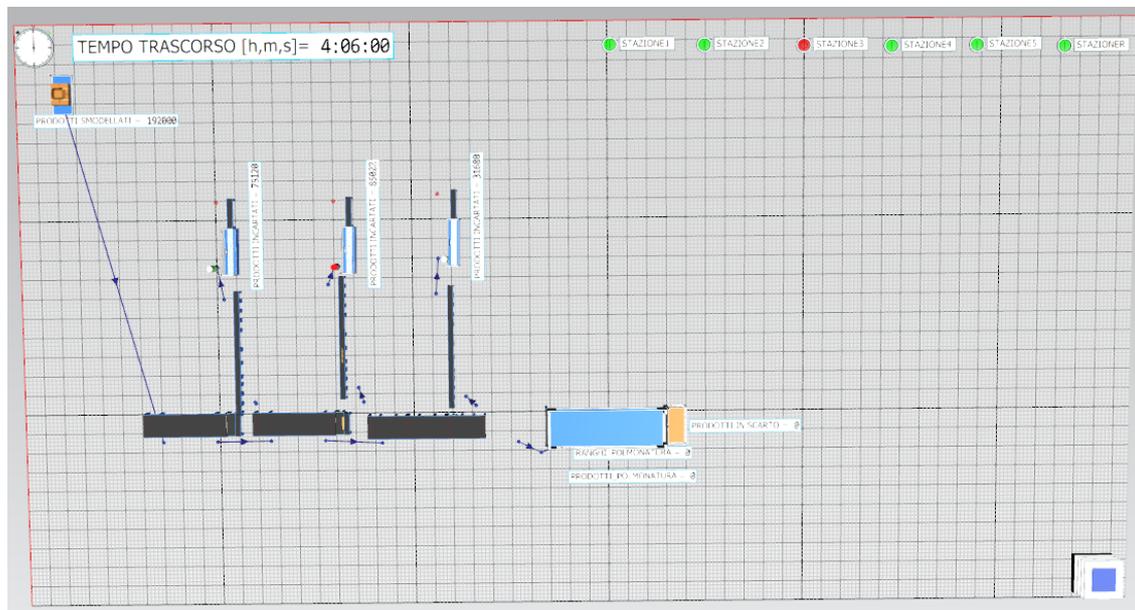


Figure 4.12 Test distribution with exit on the left

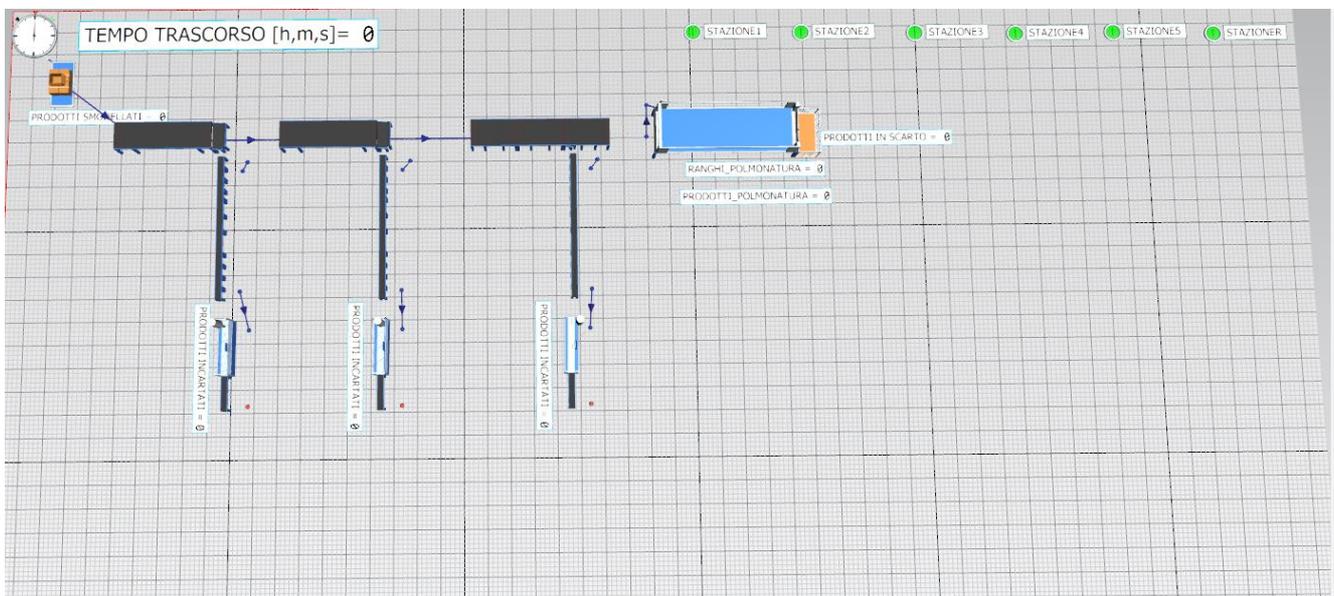


Figure 4.13 Test distribution with exit on the right

For what concerns the PIVOTSX and the PIVOTDX, the reset method also contains a section in which the initial position of the forward and backward pivots is initialized as high, in order to let any simulation start with the pivots ready to make the movement down to deliver the first product to the chart.

4.1.3 INSIGHT IN THE OBJECTS CLASSES: INIT METHOD

The **init** method has, generally, the following goals:

- Initializing the product dimensions flowing on the considered set of conveyors;
- Initializing the relevant variables of the considered object, by pointing into the specific cells of the usual table “attributi”;
- Updating the speed of the belts belonging to the considered object, in such a way that the various components of the plant are suitably fed and the wrapper can exploit its task in the best way possible.

Along with the classification just mentioned, it is possible to group the different objects according to the functions fulfilled by their init method. In depth:

- Both the **PIVOTS** and the **PIVOTD** identify the class of the predecessor of the belt belonging to the ridge: if this conveyor belongs to the class PIVOTS, then the variable called “connected_station” is exactly the inclined belt belonging to the chart of the single pivot. This method is performed in order to suitably carry out the recycling action when needed.
- In the models **LEGX**, **INTERASSEBL**, **FLOWPACKX** and **PULLNOSE** the method UpdateSpeed is called. In this method a suitable velocity to the different belts is assigned; in particular, for what concerns the legs, the following equations are considered to transfer the correct velocities to every element composing the whole leg:

$$TT3_{speed} = \frac{flowpack\ speed[\frac{pz}{min}] * product\ length}{60} * k_{TT3} \quad (4.3)$$

$$TTprec_{speed} = TTsucc_{speed} * k_{TTprec} \quad (4.4)$$

$$OR1_{speed} = TT3_{speed} * k_{OR1} \quad (4.5)$$

$$ORsucc_{speed} = ORprec_{speed} * k_{ORsucc} \quad (4.6)$$

$$ACC1_{speed} = \frac{flowpack\ speed[\frac{pz}{min}] * product\ length}{60} * k_{ACC1} \quad (4.7)$$

Where:

- $TT3_{speed}$ is the speed of TT3 conveyor;
- k_{TT3} is the corrective factor related to TT3 conveyor;
- $TTprec_{speed}$ is the speed of the conveyor preceding TT3 conveyor;
- k_{TTprec} is the corrective factor related to the conveyor preceding TT3 conveyor;
- $TTsucc_{speed}$ is the speed of the conveyor succeeding TT3 conveyor;
- $OR1_{speed}$ is the speed of OR1 conveyor;
- k_{OR1} is the corrective factor related to OR1 conveyor;
- $ORSucc_{speed}$ is the speed of the conveyor succeeding OR1 conveyor;
- k_{ORSucc} is the corrective factor related to the conveyor succeeding OR1 conveyor;
- $ORprec_{speed}$ is the speed of the conveyor preceding OR1 conveyor;
- $ACC1_{speed}$ is the speed of ACC1 conveyor;
- k_{ACC1} is the corrective factor related to ACC1 conveyor.

By considering the ACC1 as the closest to the flowpack and the ACC4 as the closest to the EPO zone, the following formulas can be derived:

$$ACCprec_{speed} = ACCsucc_{speed} * k_{ACCprec} \quad (4.8)$$

$$METAL_{speed} = ACC4_{speed} * k_{metal} \quad (4.9)$$

Where:

- $ACCprec_{speed}$ is the speed of the conveyor preceding ACC1 conveyor;
- $ACCsucc_{speed}$ is the speed of the conveyor succeeding ACC1 conveyor;
- $k_{ACCprec}$ is the corrective factor related to the conveyor preceding ACC1 conveyor;
- $METAL_{speed}$ is the speed of conveyor METAL;
- $ACC4_{speed}$ is the speed of conveyor ACC4;
- k_{metal} is the corrective factor of conveyor METAL.

Besides this speed assignment, also the initialization of the product dimensions is performed in the leg models, by assigning the correct values in the specific cells of the table “attributi”.

The same reasonings are performed for what concerns the objects INTERASSEBL, through the following formulas:

$$SuperiorBelt_{speed} = \frac{flowpack\ speed[\frac{pz}{min}] * product\ width}{60} * k_{sup} \quad (4.10)$$

$$LastInferiorBelt_{speed} = \frac{flowpack\ speed[\frac{pz}{min}] * product\ width}{60} * k_{inf} \quad (4.11)$$

$$Belt_{prec_speed} = Beltsucc_{speed} * k_{belt_{prec}} \quad (4.12)$$

Where:

- $SuperiorBelt_{speed}$ is the speed of the superior conveyor;
- k_{sup} is the corrective factor related to the superior conveyor;
- $LastInferiorBelt_{speed}$ is the speed of the last inferior conveyor;
- k_{inf} is the corrective factor related to the last inferior conveyor;
- $Belt_{prec_speed}$ is the speed of the preceding conveyor;
- $Beltsucc_{speed}$ is the speed of the succeeding conveyor;
- $k_{belt_{prec}}$ is the corrective factor related to the preceding conveyor.

Finally, both in the FLOWPACKX and in the PULLNOSE, a simple assignment of the values to the speeds of the belts is performed by pointing into the right cells of the table “attributi”. In addition, for what concerns the FLOWPACKX, the processing time of the wrapper and the speed of the conveyor located at the exit of the machine itself is given in the following way:

$$ProcTime = \frac{60}{flowpack\ speed \left[\frac{pz}{min} \right]} \quad (4.13)$$

$$ExitBelt = \frac{flowpack\ speed \left[\frac{pz}{min} \right] * LengthFlowpack * k_{exitbelt}}{60} \quad (4.14)$$

Where:

- $ProcTime$ is the time of processing of the flowpack;
- $ExitBelt$ is the speed of the exit conveyor;
- $LengthFlowpack$ is the length of the exit conveyor;
- $k_{exitbelt}$ is the corrective factor related to the exit conveyor.

In the end, in the models related to the various **buffers**, a simple initialization of the relevant variables used is performed.

4.2 Output management

A test distribution made up of two PIVOTS, one PIVOTD, three legs and flowpacks and a simple buffer, all with exit on the left, is considered:

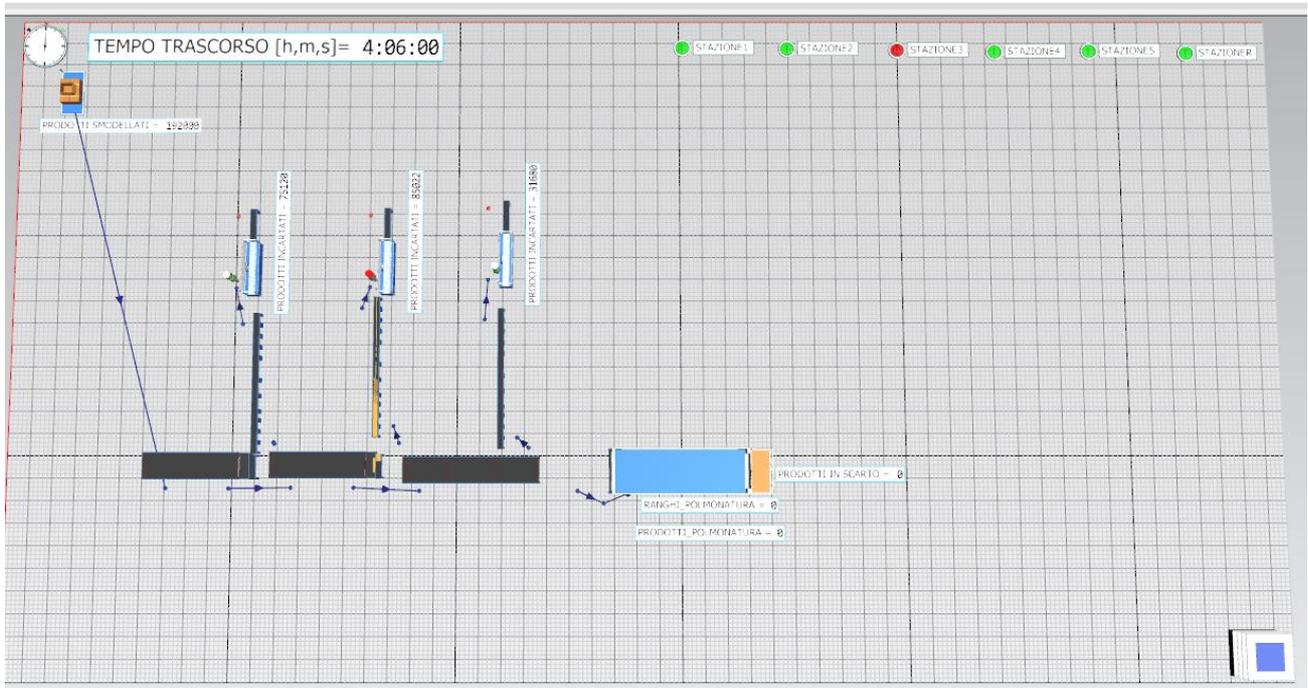


Figure 4.14 Test distribution

By looking at the frame represented in figure 4.14, it can be noticed that there are some displays along with the instances of the flowpacks and with the instances of the simple fan buffer, called BUFFERVSEMPLICE.

By starting to consider the flowpack, the display shows a value called PRODOTTI INCARTATI, that indicates the number of wrapped products. This value is obtained by

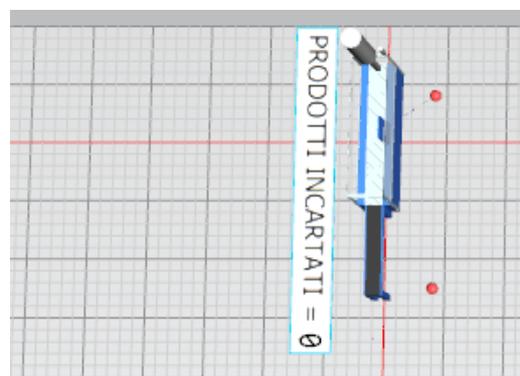


Figure 4.15 Display along with the flowpack

pointing to the drain attribute “statnumin”, which stands for the number of simple parts of product that flow into the drain itself and so that have been wrapped by the flowpack.

For what concerns the simple fan buffer **BUFFERVSEMPLICE**, the displays show three different quantities:

- **PRODOTTI IN SCARTO:** the number of products discarded, so the ones that flow into the drain when the maximum capacity of the buffer is reached and the recycling cannot be performed;
- **RANGHI_POLMONATURA:** the number of ranks contained in the buffer;
- **PRODOTTI_POLMONATURA:** the number of products contained in the buffer, obtained by multiplying the number of ranks by the number of products for each rank.

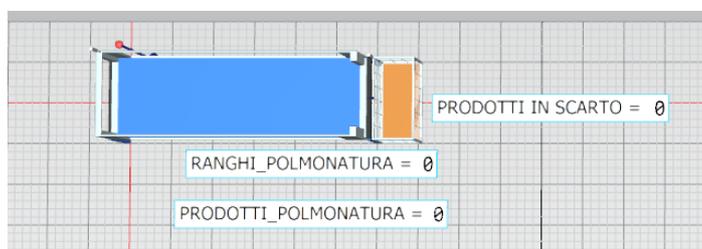


Figure 4.16 Displays in the BUFFERVSEMPLICE

In this regard, the reset methods of all the considered buffers are enriched with the definition of the variable “prod_rango”, that points to the cell of the table **PAR_GEN** in controls frame, in which **PR** (number of products per rank) is defined.

4.2.1 CONTROLS FRAME: OUTPUT PORTION

By coming back into the frame “controls”, it is possible to identify some methods and tables related to the output management. Its internal structure is now recalled in figure 4.17.

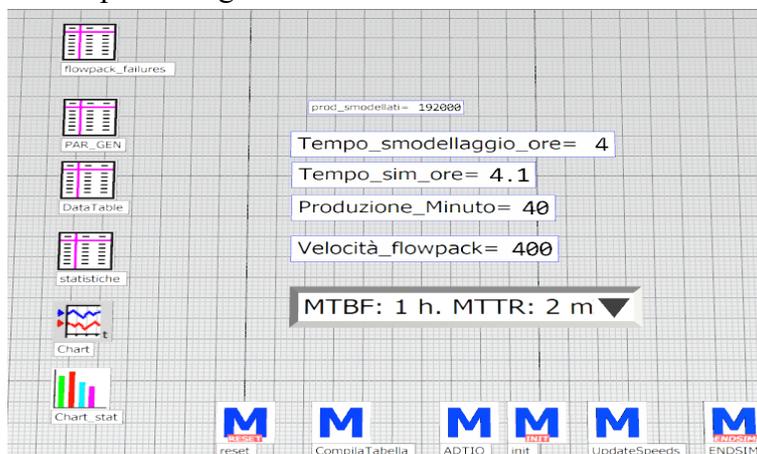


Figure 4.17 Content frame controls

As shown in figure 4.17, charts, tables and methods are present; they are related to the output management:

- The **Init** method contains instructions about the filling of a chart related to a specific variable of interest. In this case, the variable of interest taken into consideration is called “*prod_pezziora*”, it is defined in the method denominated “*stat*” and launched in the *init* method if every object is inserted in the frame, so composing the considered plant. This variable is defined as:

$$prod_{pezziora} = \frac{\text{transiting products}}{\text{simulation time}} * 3600 \quad (4.15)$$

Where:

- $prod_{pezziora}$ is the number of products produced per hour;

This variable is significant since it indicates the number of products flowing in some specific points of interest in order to identify the number of wrapped or emitted products in one hour. In fact, this variable points to different attributes according to the object considered: for what concerns the flowpacks, the variable “*prod_pezziora*” points to the number of products entered in the drain, in order to have a precise knowledge of the wrapped products, to make then a comparison with respect to the ones emitted by the source, and understand if everything worked as expected. In the case of the buffers, this variable points to the products entered in the buffer and finally, for what regards the single and double pivots and the legs, the variable considers the products flowing on a belt arbitrarily chosen, just to know how many products are flowing on the belts of the considered object.

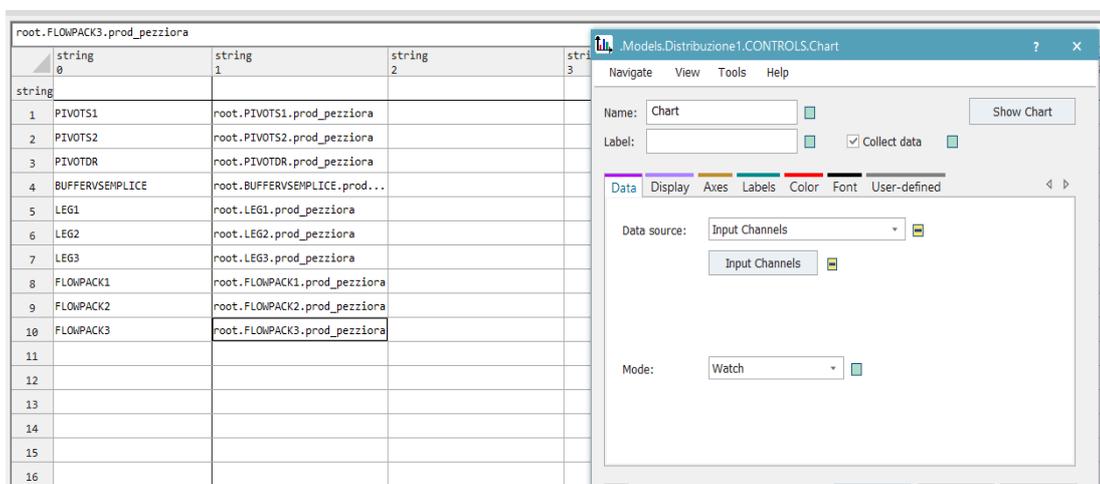


Figure 4.18 Input channels structure in chart

Figure 4.18 shows the structure of the tab input channels related to the object “chart” inserted in the frame “controls”. It can be noticed that the init method recognizes to which class every object (called node) inserted in the root frame belongs and then its path is written next to its name in the root frame. In this way, the object chart is able to plot the time history of the variable of interest.

Moreover, the object chart is inserted in every object frame, always with the aim of plotting the time history of the variable “prod_pezziora”, and while in the frame controls the init method is the main actor that plays the role of automatically filling the input channels table located in the chart, in the objects frame the variable is manually written, resulting in the layout reported in figure 4.19:

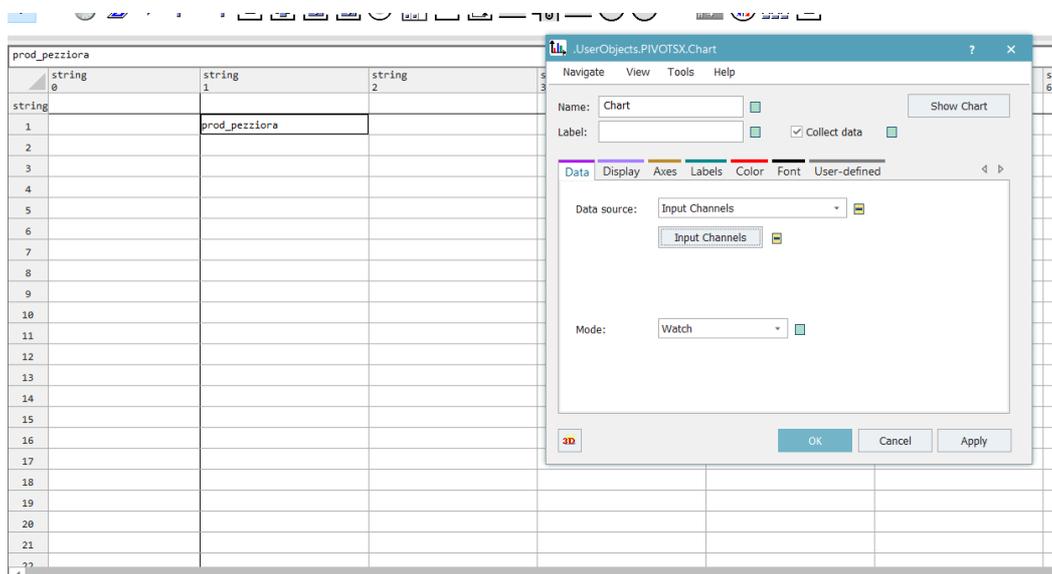


Figure 4.19 Input channels related to a PIVOTS instance

Therefore, the main difference between the object chart located in the frame controls and the ones placed in each object is that the first allows to make a comparison among the values assumed by the variable in each object of the root frame, while the objects chart located in each object only shows the variable of interest of the object itself, not allowing any comparison with the values of the same variable in all the other instances of different objects.

- The method **EndSim** is useful in order to create another interesting plot, denominated “Chart_stat”. In fact, this method allows to automatically fill the content of the table “Statistiche”, by compiling it as shown in table 4.6:

	string 0	real 1	real 2
	string name	value	
1	S_RANGHI	7420.00	
2	BUFFERVSEMPLICE	0.00	
3	FLOWPACK1	760.00	
4	FLOWPACK2	688.00	
5	FLOWPACK3	619.00	
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			

Table 4.6 Table Statistiche content

In this table the names of the elements belonging to the classes of FLOWPACKX, BUFFERVSEMPLICE, SOURCE and BUFFERGX are inserted in the first column, while the values of the products wrapped and flowing by them is reported in the second one. This table is given as input in the chart called “Chart_stat”, giving as output a histogram showing the total number of products wrapped and transited among the different objects located in the root frame. Figure 4.2 shows the way in which the table Statistiche feeds the specific tab of “Chart_stat” (highlighted in blue):

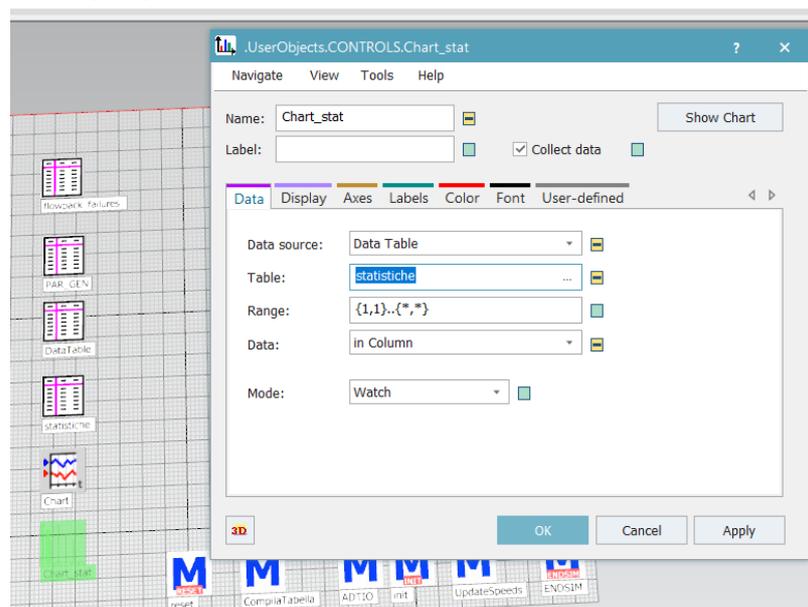


Figure 4.20 Table Statistiche as input of Chart_stat

4.3 Failures management

The output data and statistics cannot be considered reliable if no failures in the machines are taken into account. In particular, the failures are considered in the flowpacks activities, therefore in the wrapping machines and also in the buffers, by taking into consideration how many products are discarded if the flowpacks are not working as expected or if the plant is oversized.

In these regards, both the frames of the objects FLOWPACKX, BUFFERVSEMPLICE (and also BUFFERGX) are deeply analysed.

4.3.1 INSIGHT IN THE FLOWPACKX

The FLOWPACKX frame results to be rich of methods, charts and tables, as shown in figure 4.21; they are going to be analysed one by one in order to clarify their utility.

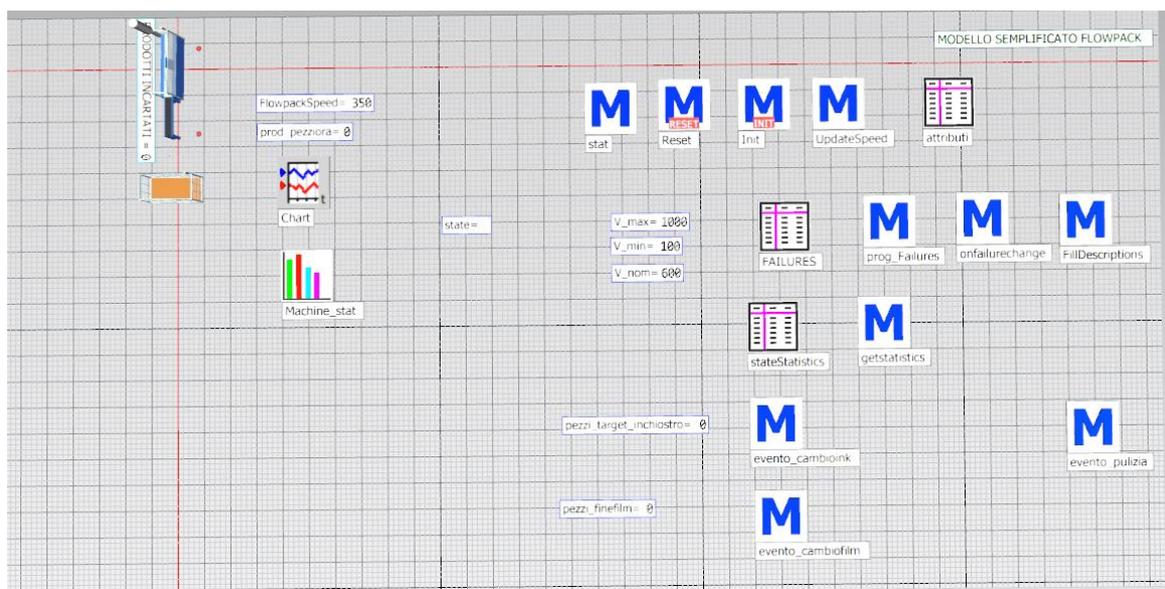


Figure 4.21 FLOWPACKX frame structure

It must be noticed that in the frame are still present some methods and tables that have been used in the past as temporary solution, but now are discarded since further improvements have been introduced. For these reasons, the methods “onfailurechange”, “FillDescriptions” and the table “FAILURES” will not be illustrated in the upcoming section.

- In the method “**prog_Failures**”, the table denoted as “flowpack_failures” belonging to the frame “controls” is called. This table contains the definition of five different failures belonging to the flowpacks and the definition of the MTBF (mean time before failure), which is the time passed before a failure occurs (expressed in hours) and the MTTR

(mean time to repair), which is the time needed to repair the failure. Moreover, another column related to the activation or not of the considered failure is present. In figure 4.7 the just described structure is shown:

0	string	boolean	string	string	string	string	string
Failures	Active	MTBF [h]	MTTR [min]				
1 failure_1	true	2	5				
2 failure_2	false						
3 failure_3	false						
4 failure_4	false						
5 failure_5	false						
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							

Table 4.7 Flowpack_failures content

The range of values that can be assigned as MTBF goes from 1h to 24h, while the range of values for the MTTR goes from 1min to 60min; these numbers can be selected by means of a drop-down menu.

Then, all these failures are inserted as possible options in the tab present in the flowpack object related to the failures, resulting in the structure shown in figure 4.22:

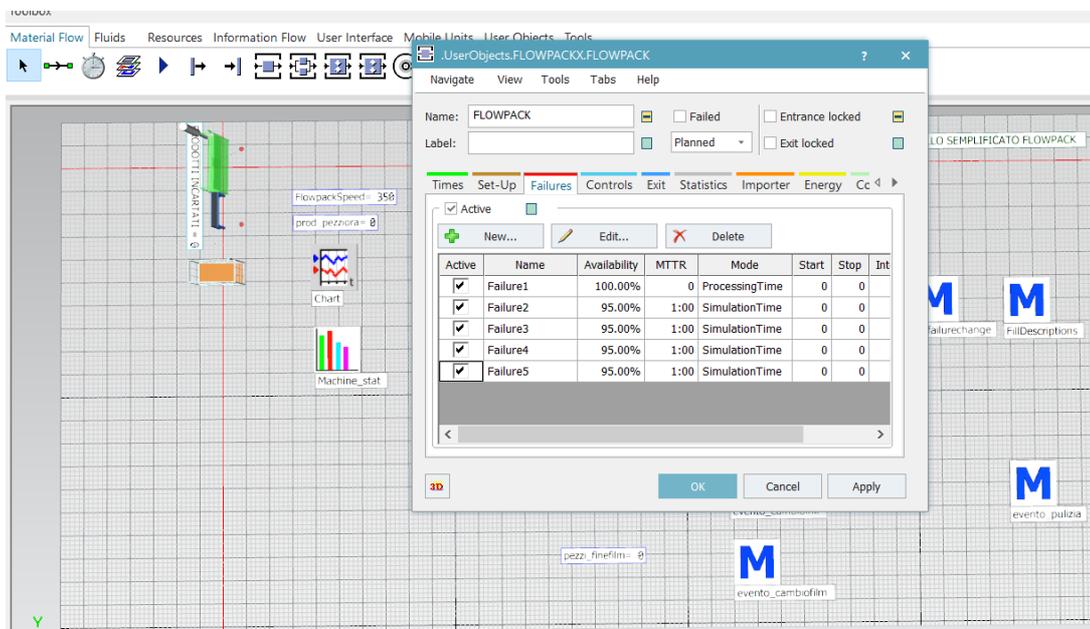


Figure 4.22 Tab failures in the flowpack

The method “prog_Failures” fulfils the following task: it checks in the table “flowpack_failures” if the failure considered is active or not; then it activates or not the corresponding failure in the tab failures belonging to the considered flowpack and sets the corresponding MTBF as interval and MTTR as duration times expressed in seconds. All these failures set in the method “prog_Failures” and consequently in the tab failures of the flowpacks are stochastic failures, modelled by means of a negative exponential distribution, which is specified in the method.

- Once these failures have been suitably programmed and assigned to the different machines, the method “getstatistics” is used in order to have a precise knowledge of the percentage of usage of the different wrappers. With this code, in fact, the table “statestatistics” is filled with the percentage of working condition, fail and waiting condition is reported table 4.8:

	string 0	real 1
string	state	percentage
1	working	93.96
2	failed	0.00
3	waiting	6.04
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
--		

**Table 4.8 Table "statestatistics"
content**

Then, this table is given as input to a chart called “Machine_stat”, in which a graph related to the percentages of usage, failure and waiting is built. Figure 4.23 shows the tab of the chart “Machine_stat” in which the table “statestatistics” is fed in input.

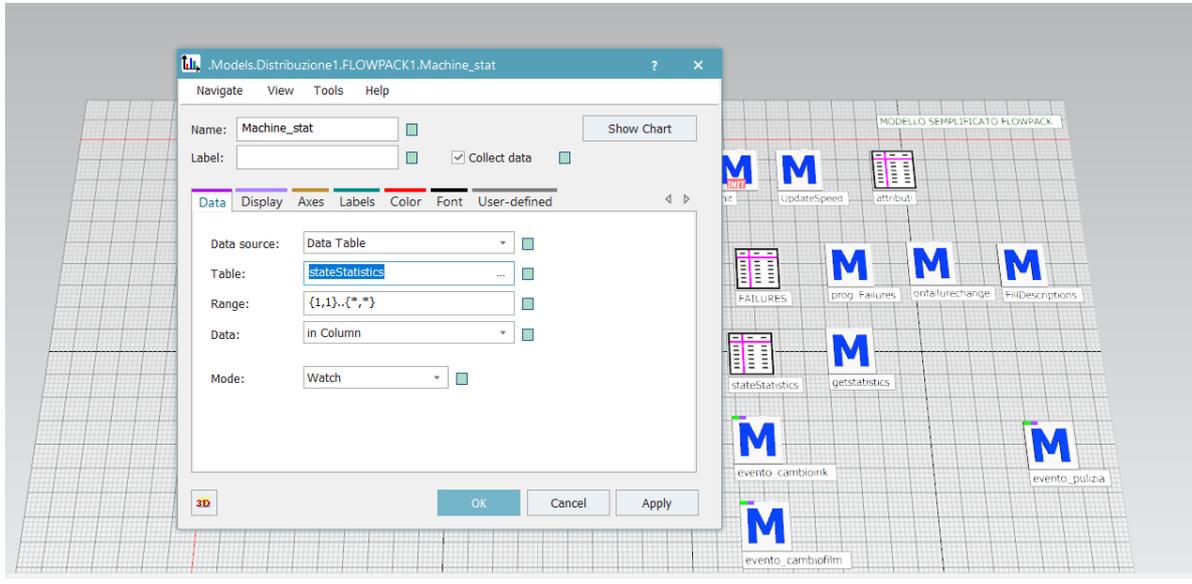


Figure 4.23 "Statestatistics" as input to "Machine_stat"

- Finally, three different methods are created in order to deal with deterministic failures, i.e, failures that occur at precise instants, for a precise duration. The three methods are: “evento_cambioink”, “evento_pulizia” and “evento_cambiofilm”. The first one is related to the change of the ink cartridge to write on the shell; the second one is related to the cleaning of the belts and the last one is related to the change of the paper web that wraps the products.

The method “evento_cambioink” is based on the usage of the variable “pezzi_target_inchiostro”. In particular, this variable is incremented every time a product enters the drain, in order to keep track of the products wrapped; the method points to specific data in the table “attribute” where a certain number of pieces after which the change of the ink cartridge must be performed is defined, along with a time indicating the duration of this procedure. So when the value of the variable “pezzi_target_inchiostro” is equal to the value inserted in the table, the machine stops for the duration of the change operation and when it restarts, the variable is reset to 0.

The second method “evento_pulizia”, instead, works by performing a cleaning operation at every interval established as a variable in the table “attributi” and for a duration always defined in the same table, therefore blocking the flowpack functioning for such a period. Once the cleaning is performed, it is carried out again exactly after the same quantity of

time provided by the value of the interval, not starting the count after the duration time needed to finish the operation.

Finally, the method “evento_cambiofilm” exploits the usage of the variable “pezzi_finefilm” and works in the very same way of the first method, by blocking the normal functioning of the flowpack when the target number of products is reached and then resetting to zero this variable once the change has been performed for the duration provided by the specified value in the table “attributi”.

All the methods illustrated, either the ones related to the stochastic failures and the ones related to the deterministic events, are called in the init method of the FLOWPACKX frame, in order to let them execute at the beginning of the simulation. Then, each of them recalls itself after a predefined amount of time, in order to always refresh the incoming and outgoing values.

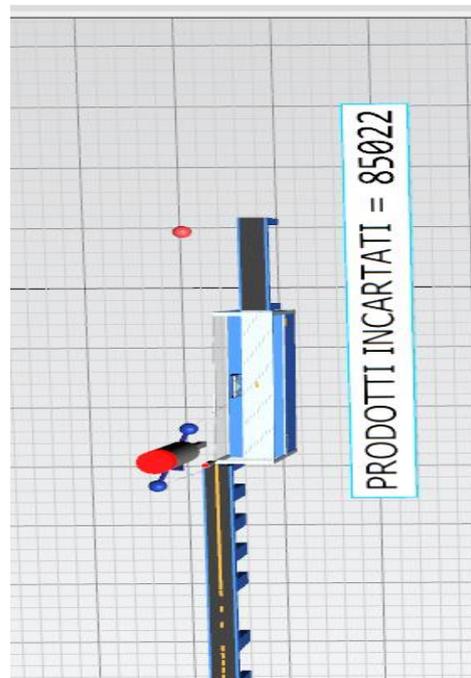


Figure 4.24 Flowpack tower colour when a failure occurs

4.3.2 INSIGHT IN THE BUFFERS

Both the BUFFERVSEMPLICE and the BUFFERGX frames are structured in the same way, for this reason, only the BUFFERVSEMPLICE is considered its structure is reported in figure 4.25:

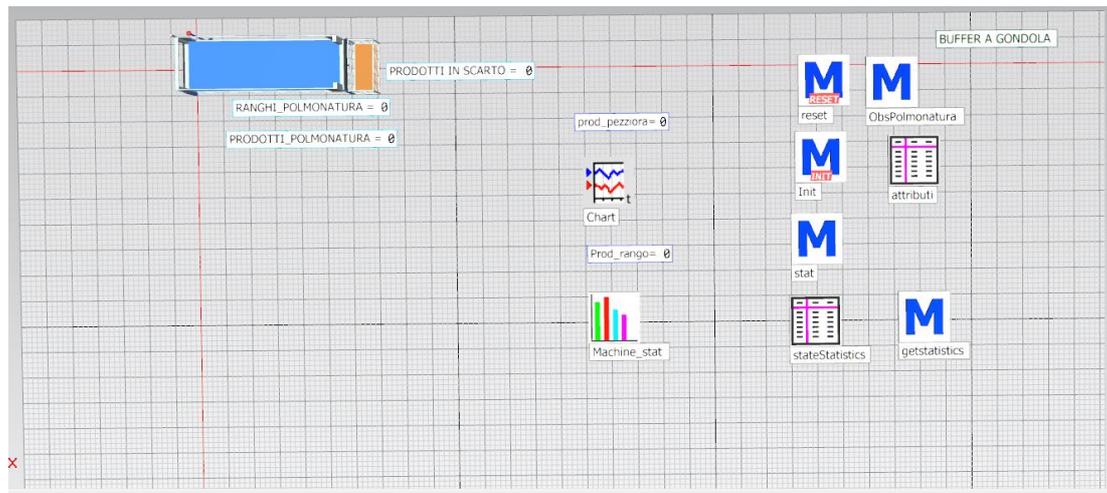


Figure 4.25 BUFFERVSEMPLICE frame structure

The failures are handled by the “**getstatistics**” method. The difference between this method and the other denominated in the same way stands in the attributes to which this method points.

In particular, the attributes of interest in this method are:

- **StatRelativeEmptyPortion:** returns in percentage the relative portion of the statistics collection period during which the object was empty with respect to the time during which the object was available;
- **Statnumin:** returns the number of entries in the buffer;
- **Statrelativeoccupation:** returns in percentage the relative occupancy without any interruption of the buffer [16].

These values are automatically written in the table “stateStatistics”, which is filled by the described method. Finally, this table is fed as input to the chart called “Machine_stat” that returns a graph representing the variables previously illustrated.

Chapter 5 - Tests and analytical results

Different tests are performed on the two kinds of plants taken into consideration in order to evaluate how they should work in the real environment. Successively, further considerations are performed on the accuracy of the models developed in the virtual environment.

5.1 Test on the first distribution

The first test is performed on a distribution composed of two PIVOTS, one PIVOTD, three legs and flowpacks and a simple buffer, all with exit on the left, as reported in figure 5.1:

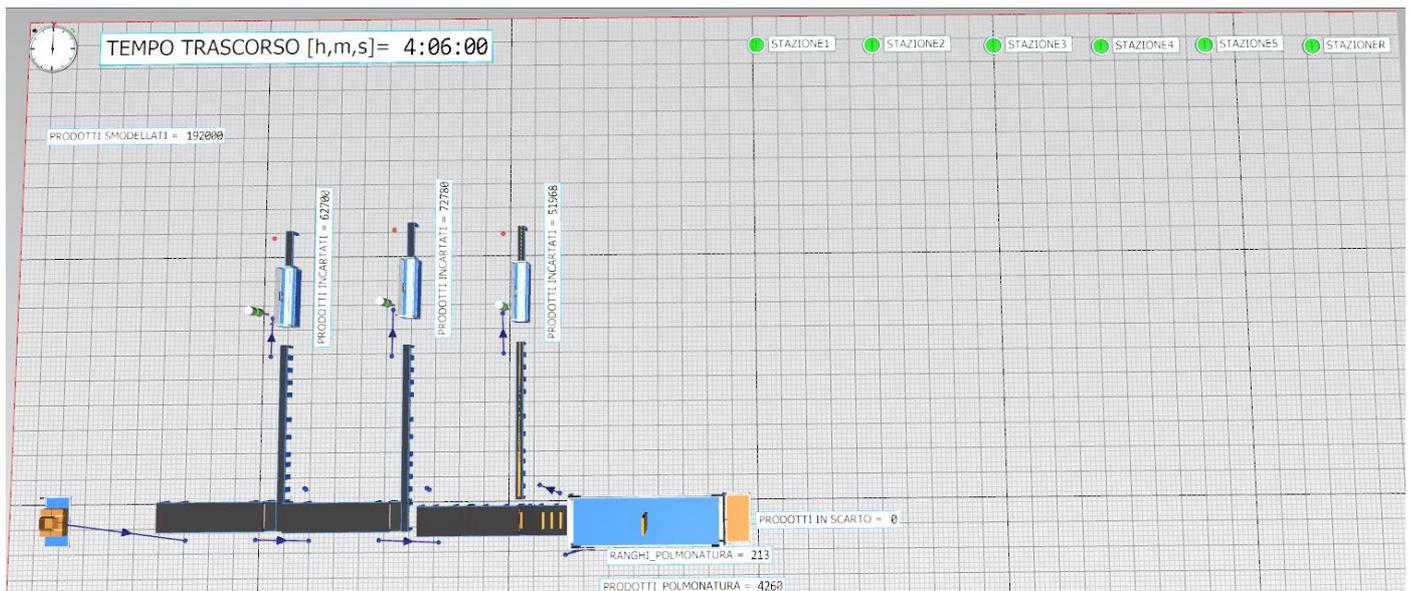


Figure 5.1 First test distribution

The input parameters are set as reported in Tables 5.1 and 5.2.

Tempo_Smodellaggio_Ore =	4
Tempo_Sim_Ore =	4,1
Produzione_Minuto =	40
Velocità_Flowpack =	400

Table 5.1 Input data in controls frame

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
1	RISERVA_1							prodotto			
2	RISERVA_2							prodotto			
3	RISERVA_3							prodotto			
4	L_LunghezzaProdotti	90						prodotto			
5	W_LarghezzaProdotti	30						prodotto			
6	H_AltezzaProdotti	11.7						prodotto			
7	P_DistanzaProdotti	2.5						prodotto			
8	R_DistanzaRanghi	5						prodotto			
9	M_DistanzaStampi	0						prodotto			
10	PR_NumeroProdotti					20		prodotto			
11	RM_NumeroRanghi					1		prodotto			
12	RISERVA_12							prodotto			
13	RISERVA_13							prodotto			
14	RISERVA_14							prodotto			
15	RISERVA_15							prodotto			
16	RISERVA_16							prodotto			
17	MUwidth	647.5						prodotto			
18	MULength	90						prodotto			
19	MUHeight	11.7						prodotto			
20	RISERVA_20							prodotto			
21	RISERVA_21							prodotto			
22	RISERVA_22							prodotto			
23	RISERVA_23							prodotto			

Table 5.2 PAR_GEN input data for mobile units

With the assigned data, the source will stop emitting products after 4 hours, while the simulation is going to stop after 4h 06'. Moreover, the variable "Produzione_minuto" is computed as:

$$\begin{aligned}
 \text{Produzione minuto} &= \frac{\text{Velocità flowpack} * \text{number of flowpack to be fed}}{PR * RM} \\
 &= \frac{400 * 2}{20 * 1} = 40 \left[\frac{\text{mobile units}}{\text{min}} \right]
 \end{aligned}$$

This value is obtained in order to saturate two out of three flowpacks and use the third one as a jolly flowpack.

The flowpack velocity, as already shown in the previous formula, is fixed:

$$\text{Velocità flowpack} = 400 \left[\frac{\text{pz}}{\text{min}} \right]$$

At the end of the simulation, the displays located on the root frame show the following values:

- **PRODOTTI SMOCELLATI: 192000**
- **PRODOTTI INCARTATI FLOWPACK1: 62700**
- **PRODOTTI INCARTATI FLOWPACK2: 72780**
- **PRODOTTI INCARTATI FLOWPACK3: 51968**
- **RANGHI_POLMONATURA: 213**
- **PRODOTTI_POLMONATURA: 4260**
- **PRODOTTI IN SCARTO: 0**

These results are obtained by considering the failures profile in tables 5.3, 5.4, 5.5, 5.6:

	string 0	boolean 1	string 2	string 3	st 4
	Failures	Active	MTBF [h]	MTTR [min]	
1	failure1	true	2	4	
2	failure2	false	1	2	
3	failure3	false	1	2	
4	failure4	false	1	2	
5	failure5	false	1	2	
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

Table 5.3 Stochastic failures profile

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
17	Pezzi_Stampa					12000		FLOWPACK			
18	T_Cambio_inchiostro							FLOWPACK	10:00.0000		
19	Pezzi_fine_film					24000		FLOWPACK			
20	T_Cambio_film							FLOWPACK	5:00.0000		
21	RISERVA							FLOWPACK			
22	RISERVA							FLOWPACK			
23	RISERVA							FLOWPACK			
24	RISERVA							FLOWPACK			
25	RISERVA							FLOWPACK			
26	RISERVA							FLOWPACK			
27	RISERVA							FLOWPACK			
28	RISERVA							FLOWPACK			
29	RISERVA							FLOWPACK			
30	RISERVA							FLOWPACK			
31	RISERVA							FLOWPACK			
32	RISERVA							FLOWPACK			
33	intervallo_pulizia							FLOWPACK	8:00:00.0000		
34	durata_pulizia							FLOWPACK	10:00.0000		
35	RISERVA							FLOWPACK			
36	RISERVA							FLOWPACK			
37	uscita_strappator...						false	FLOWPACK			
38	RISERVA							FLOWPACK			

Table 5.4 Flowpack1 deterministic failures

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
17	Pezzi_Stampa					16000		FLOWPACK			
18	T_Cambio_inchiostro							FLOWPACK	10:00.0000		
19	Pezzi_fine_film					32000		FLOWPACK			
20	T_Cambio_film							FLOWPACK	5:00.0000		
21	RISERVA							FLOWPACK			
22	RISERVA							FLOWPACK			
23	RISERVA							FLOWPACK			
24	RISERVA							FLOWPACK			
25	RISERVA							FLOWPACK			
26	RISERVA							FLOWPACK			
27	RISERVA							FLOWPACK			
28	RISERVA							FLOWPACK			
29	RISERVA							FLOWPACK			
30	RISERVA							FLOWPACK			
31	RISERVA							FLOWPACK			
32	RISERVA							FLOWPACK			
33	intervallo_pulizia							FLOWPACK	8:00:00.0000		
34	durata_pulizia							FLOWPACK	1:00:00.0000		
35	RISERVA							FLOWPACK			
36	RISERVA							FLOWPACK			
37	uscita_strappator...						false	FLOWPACK			
38	RISERVA							FLOWPACK			

Table 5.5 Flowpack2 deterministic failures

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
	stringName of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
17	Pezzi_Stampa					20000		FLOWPACK			
18	T_Cambio_inchiostro							FLOWPACK	10:00.0000		
19	Pezzi_fine_film					36000		FLOWPACK			
20	T_Cambio_film							FLOWPACK	5:00.0000		
21	RISERVA							FLOWPACK			
22	RISERVA							FLOWPACK			
23	RISERVA							FLOWPACK			
24	RISERVA							FLOWPACK			
25	RISERVA							FLOWPACK			
26	RISERVA							FLOWPACK			
27	RISERVA							FLOWPACK			
28	RISERVA							FLOWPACK			
29	RISERVA							FLOWPACK			
30	RISERVA							FLOWPACK			
31	RISERVA							FLOWPACK			
32	RISERVA							FLOWPACK			
33	intervallo_pulizia							FLOWPACK	8:00:00.0000		
34	durata_pulizia							FLOWPACK	1:00:00.0000		
35	RISERVA							FLOWPACK			
36	RISERVA							FLOWPACK			
37	uscita_strappator...						false	FLOWPACK			
38	RISERVA							FLOWPACK			
...	RISERVA							FLOWPACK			

Table 5.6 Flowpack3 deterministic failures

It can be noticed that the two different deterministic failures related to the change of the ink and to the change of the film are set in order to happen in different moments on the different flowpacks, considering that:

$$\begin{aligned} \text{theoretical flowpacked prod} &= \text{Velocità_flowpack} \left[\frac{\text{pz}}{\text{min}} \right] * 60[\text{min}] = 400 * 60 \\ &= 24000 \left[\frac{\text{pz}}{\text{h}} \right] \end{aligned}$$

For this reason, the numbers set in the previous tables are chosen according to this value, in order to let the failures suitably appear in a specific moment and then periodically occur again.

Moreover, for what concerns the cleaning, it is set to be equal for all the different machines and it should be performed after 8 hours, so this kind of deterministic event is not considered in the following test.

The number of emitted products from the source is coherent, since it is given by the following relationship:

$$\begin{aligned} \text{modelled prod} &= PR * RM * \text{Produzione minuto} * 60 * \text{Tempo smodellaggio ore} \\ &= 20 * 1 * 40 * 60 * 4 = 192000 [\text{pz}] \end{aligned}$$

Furthermore, it can be noticed that the wrapped products by the first two flowpacks are more than the ones wrapped by the third: this is due to the fact that the plant is sized in order to make only two flowpacks work and use the third one as a jolly, so the latter is used only

when the failures affecting the first two flowpacks do not make them properly fulfil their task.

In addition, the drain contains no product, because the buffer discards products in the drain connected to it only when the line is oversized and the flowpacks are not able to wrap all the incoming products.

The chart related to the variable “prod_pezziora” referred to any object present in the root frame is reported in figure 5.2, 5.3:

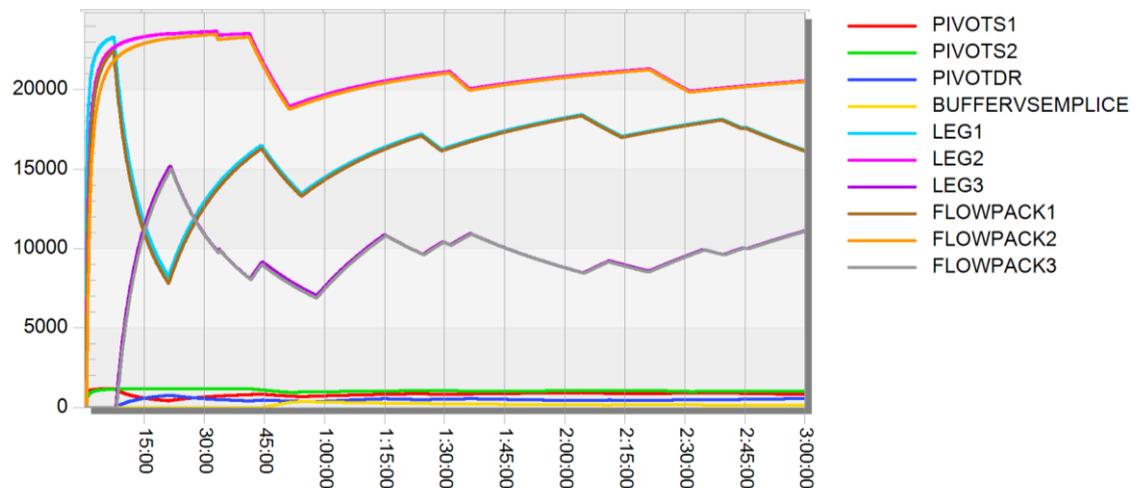


Figure 5.2 prod_pezziora plot for the first three hours

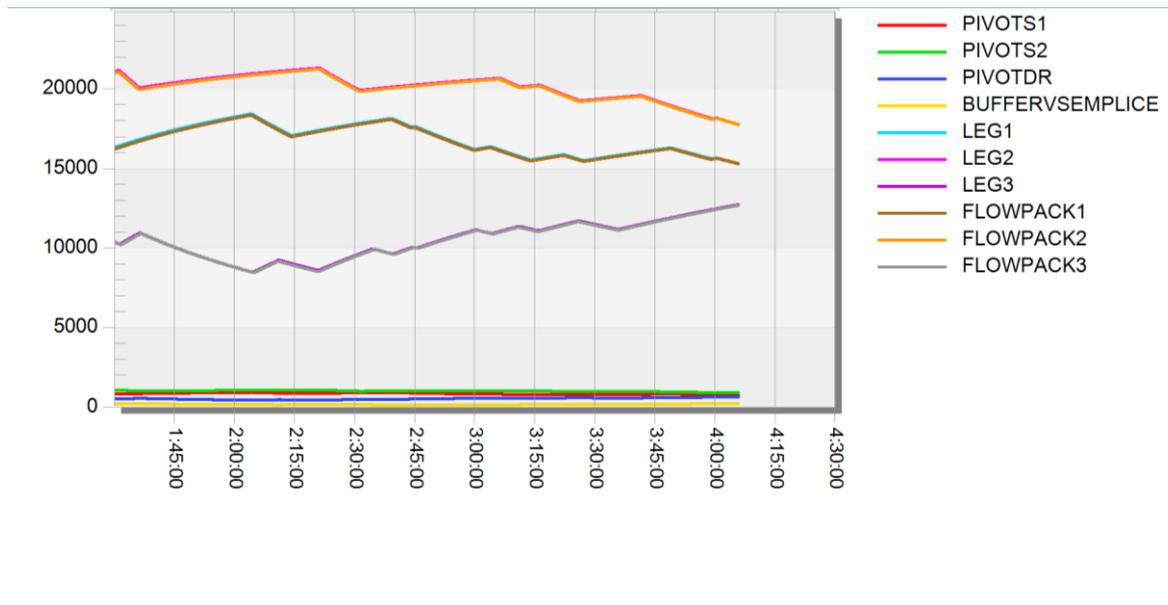


Figure 5.3 Plot of prod_pezziora till the end of simulation

The behaviour of these lines indicates the production per hour and it is coherent with the numbers reported in the displays, since it can be easily seen that the flowpack 1 has a value that lays around 15500 [pz/h] and in fact:

$$prod\ pezziora\ flowp1 = \frac{62700}{4} = 15675 \left[\frac{pz}{h} \right]$$

The same reasonings hold for the other 2 flowpacks. Moreover, it can be noticed that the lines related to each flowpack are overlapped on the ones related to each leg: this is due to the fact that the products that are flowing in a specific leg are then flowing in the connected flowpack.

For what concerns the graph “Chart_stat”, the obtained histogram is the one reported in figure 5.4:

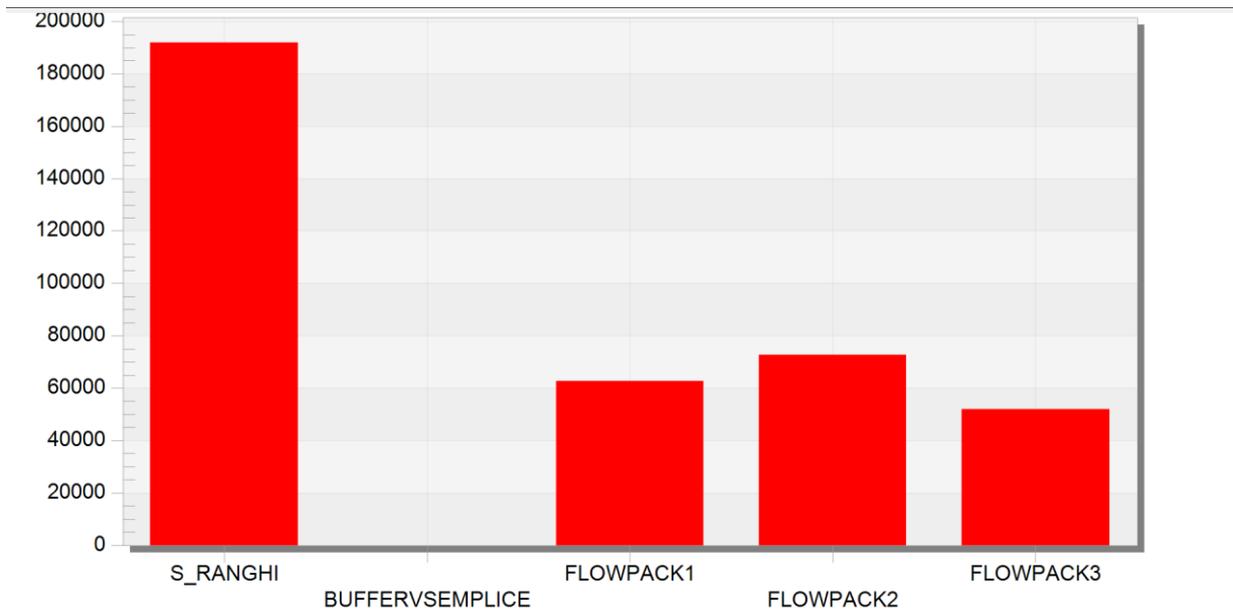


Figure 5.4 Chart_stat histogram

Also in this case, the histogram realistically represents the situation, since the height of the different columns exactly coincide with the values assumed by the displays located in the root frame, either for the flowpacks and for the simple fan buffer.

Finally, the three graphs related to the percentage of usage of the three flowpacks are reported in figure 5.5, 5.6, 5.7.



Figure 5.5 Flowpack1 percentage of usage



Figure 5.6 Flowpack2 percentage of usage



Figure 5.7 Flowpack3 percentage of usage

It is evident that the third wrapper works less in percentage than the first two, for the reasons already explained. In addition, the flowpack 3 is the one that lays more than any other flowpack in waiting, since the products seldom arrive to this wrapper for the sizing of the plant. The flowpack 1, instead, is the one with the highest percentage of failure, since the data set for the deterministic failures are the lowest, causing a higher occurrence of these kind of failures in addition to the stochastic ones.

It must be also underlined that the deterministic events are not plotted as the stochastic failures, but they are considered by means of the final value of the variable shown by the display “PRODOTTI INCARTATI”, because when the events take place, the entrance of the affected flowpack is locked, its tower assumes the red colour as for the stochastic failures and so the display stops increasing its value till the end of the deterministic event.

In the end, the graphs related to the simple fan buffer result are the one reported in figure 5.8:

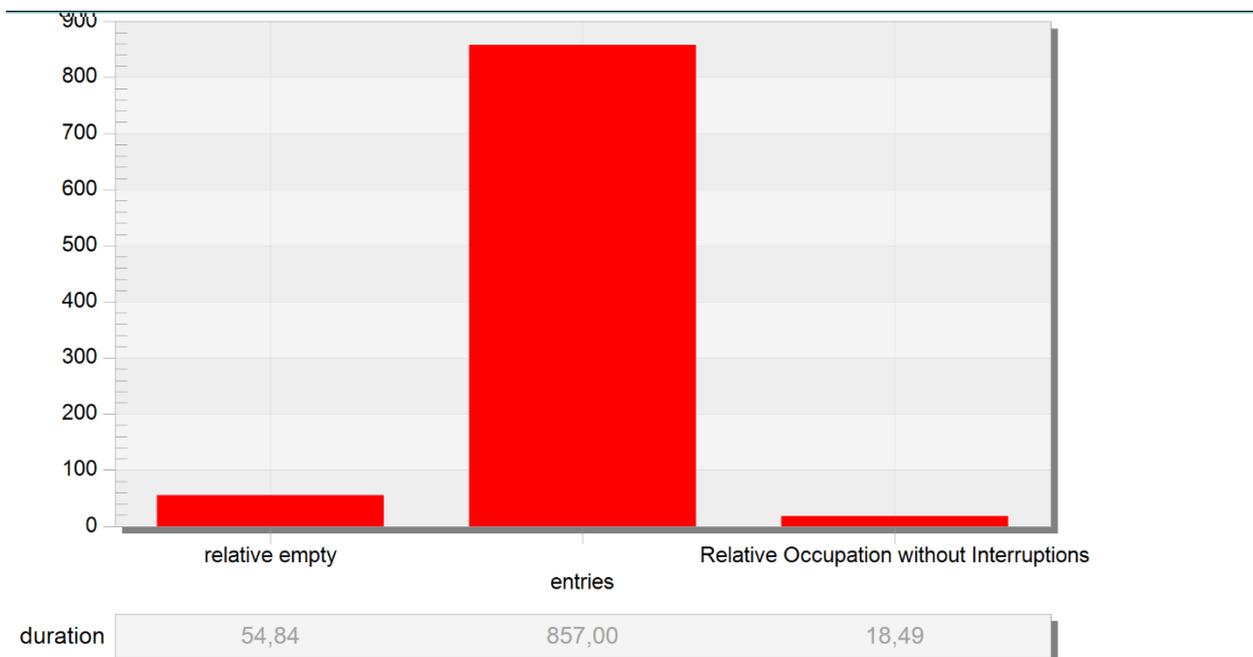


Figure 5.8 Simple fan buffer statistics

This statistic shows that for almost the 55% of the simulation time, the buffer results to be empty and this is coherent with the plant dimensioning. The relative occupation with no interruptions is almost 19%, putting in evidence that many failures affecting the wrappers often occur interrupting the filling of the buffer. The entries are not reported in terms of percentage but in terms of absolute number and they are exactly 857: so the buffer has been crucial in order to keep some products in it when also the flowpack 3 has been affected by some failures; then, the products stored in it have been recirculated on the flowpack 3 in the first proper moment.

Finally, the efficiency related to the different wrappers and to the whole plant is so computed as:

$$\text{Flowpack1 efficiency} = \frac{\text{working time}}{\text{working time} + \text{failed time}} = \frac{63.98}{63.98 + 34.04} = 0.65$$

$$\text{Flowpack2 efficiency} = \frac{\text{working time}}{\text{working time} + \text{failed time}} = \frac{74.27}{74.27 + 23.73} = 0.76$$

$$\text{Flowpack3 efficiency} = \frac{\text{working time}}{\text{working time} + \text{failed time}} = \frac{52.7}{52.7 + 15.79} = 0.77$$

$$\text{Plant efficiency} = \frac{\sum \text{prodotti incartati}}{\text{prodotti smodellati}} = \frac{62700 + 72780 + 51968}{192000} = 0.98$$

For the sake of simplicity, these computations are made by neglecting the products still laying on the line when the simulation stops, given by in the products that the buffer is recirculating on the third leg and flowpack.

5.2 Test on the second distribution

The second test is performed on a distribution composed of a PIVOTSX, a PULLNOSE, a BUFFERG, a INTERASSEBL and three LEGX connected to three different FLOWPACKX, as shown in figure 5.9:

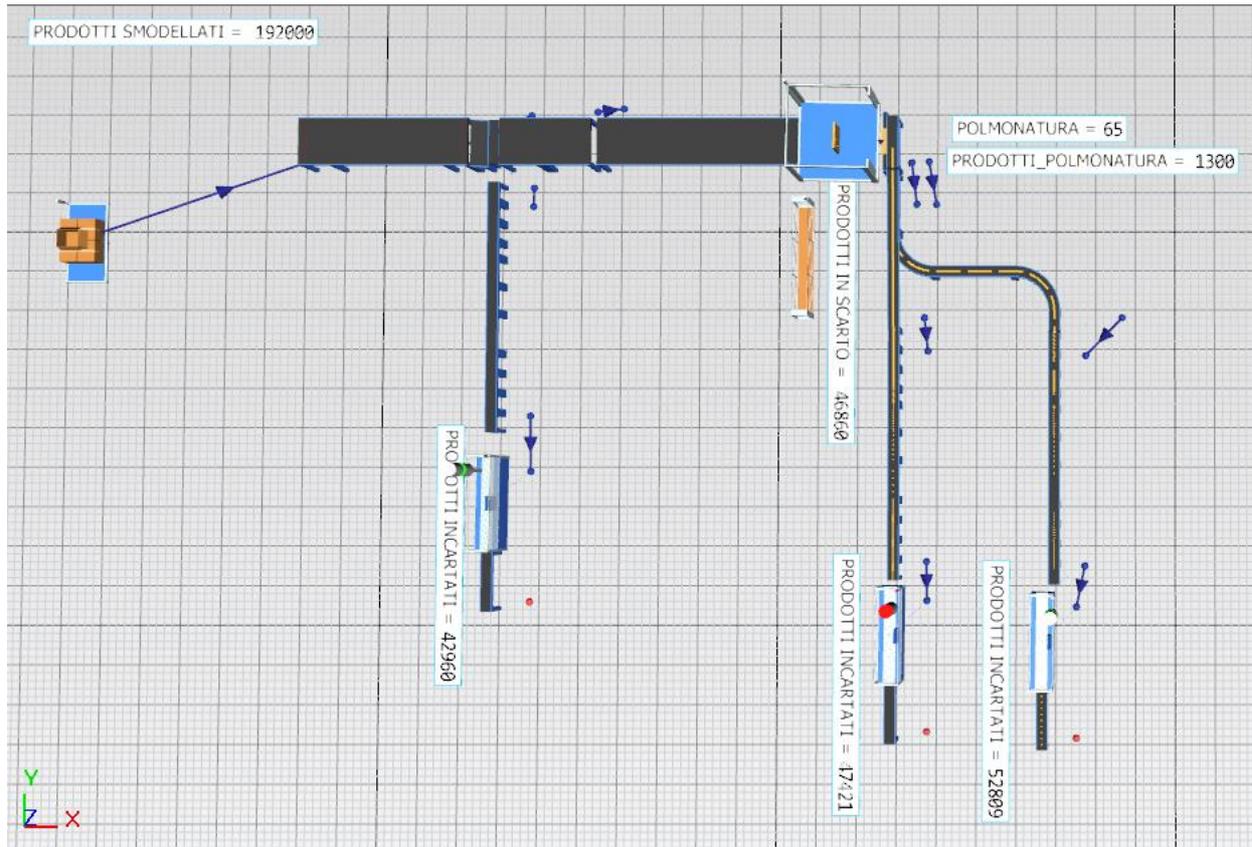


Figure 5.9 Second test distribution

The input parameters are set as shown in tables 5.7, 5.8.

Tempo_Smodellaggio_Ore =	2
Tempo_Sim_Ore =	2,2
Produzione_Minuto =	20
Velocità_Flowpack =	600

Table 5.7 Input data in controls frame

1	string	length[mm]	speed[m/min]	acceleratio	real	integer	boolean	string	time	date	datetime
2	2	3	4	5	6	7	8	9	10	11	
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
1	RISERVA_1							prodotto			
2	RISERVA_2							prodotto			
3	RISERVA_3							prodotto			
4	L_LunghezzaProdotti	60						prodotto			
5	W_LarghezzaProdotti	30						prodotto			
6	H_AltezzaProdotti	11.7						prodotto			
7	P_DistanzaProdotti	2.5						prodotto			
8	R_DistanzaRanghi	5						prodotto			
9	M_DistanzaStampi	0						prodotto			
10	PR_NumeroProdotti					20		prodotto			
11	RM_NumeroRanghi					4		prodotto			
12	RISERVA_12							prodotto			
13	RISERVA_13							prodotto			
14	RISERVA_14							prodotto			
15	RISERVA_15							prodotto			
16	RISERVA_16							prodotto			
17	MUWidth	647.5						prodotto			
18	MULength	255						prodotto			
19	MUHeight	11.7						prodotto			
20	RISERVA_20							prodotto			
21	RISERVA_21							prodotto			
22	RISERVA_22							prodotto			

Table 5.8 Input data in PAR_GEN

With the assigned data, the source will stop emitting products after 2 hours, while the simulation is going to stop after 2h 12'. Moreover, the variable "Produzione_minuto" is computed as:

$$\begin{aligned}
 \text{Produzione minuto} &= \frac{\text{Velocità flowpack} * \text{number of flowpack to be fed}}{PR * RM} \\
 &= \frac{600 * 2}{20 * 4} = 15 \left[\frac{\text{mobile units}}{\text{min}} \right]
 \end{aligned}$$

This variable is chosen equal to 20 instead, in order to let the third flowpack and the drain of the buffer work.

The flowpack velocity is fixed:

$$\text{Velocità flowpack} = 600 \left[\frac{\text{pz}}{\text{min}} \right]$$

At the end of the simulation, the displays located on the root frame show the following values:

- **PRODOTTI SMOCELLATI:** 192000
- **PRODOTTI INCARTATI FLOWPACK1:** 42960
- **PRODOTTI INCARTATI FLOWPACK2:** 47421
- **PRODOTTI INCARTATI FLOWPACK3:** 52809
- **RANGHI_POLMONATURA:** 65
- **PRODOTTI_POLMONATURA:** 1300
- **PRODOTTI IN SCARTO:** 46860

These results are obtained by the failures profile shown in tables 5.9, 5.10, 5.11, 5.12:

	string 0	boolean 1	string 2	string 3	st 4
string	Failures	Active	MTBF [h]	MTTR [min]	
1	failure1	true	2	4	
2	failure2	false	1	2	
3	failure3	false	1	2	
4	failure4	false	1	2	
5	failure5	false	1	2	
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

Table 5.9 Stochastic failures profile

V_max											
	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
17	Pezzi_Stampa					10000		FLOWPACK			
18	T_Cambio_inchiostro							FLOWPACK	10:00.0000		
19	Pezzi_fine_film					20000		FLOWPACK			
20	T_Cambio_film							FLOWPACK	5:00.0000		
21	RISERVA							FLOWPACK			
22	RISERVA							FLOWPACK			
23	RISERVA							FLOWPACK			
24	RISERVA							FLOWPACK			
25	RISERVA							FLOWPACK			
26	RISERVA							FLOWPACK			
27	RISERVA							FLOWPACK			
28	RISERVA							FLOWPACK			
29	RISERVA							FLOWPACK			
30	RISERVA							FLOWPACK			
31	RISERVA							FLOWPACK			
32	RISERVA							FLOWPACK			
33	intervallo_pulizia							FLOWPACK	2:00:00.0000		
34	durata_pulizia							FLOWPACK	1:00:00.0000		
35	RISERVA							FLOWPACK			
36	RISERVA							FLOWPACK			
37	uscita_strappator...						true	FLOWPACK			
38	RISERVA							FLOWPACK			
..	RISERVA							FLOWPACK			

Table 5.10 Flowpack1 deterministic failures

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
17	Pezzi_Stampa					15000		FLOWPACK			
18	T_Cambio_inchiostro							FLOWPACK	10:00.0000		
19	Pezzi_fine_film					30000		FLOWPACK			
20	T_Cambio_film							FLOWPACK	5:00.0000		
21	RISERVA							FLOWPACK			
22	RISERVA							FLOWPACK			
23	RISERVA							FLOWPACK			
24	RISERVA							FLOWPACK			
25	RISERVA							FLOWPACK			
26	RISERVA							FLOWPACK			
27	RISERVA							FLOWPACK			
28	RISERVA							FLOWPACK			
29	RISERVA							FLOWPACK			
30	RISERVA							FLOWPACK			
31	RISERVA							FLOWPACK			
32	RISERVA							FLOWPACK			
33	intervallo_pulizia							FLOWPACK	2:00:00.0000		
34	durata_pulizia							FLOWPACK	1:00:00.0000		
35	RISERVA							FLOWPACK			
36	RISERVA							FLOWPACK			
37	uscita_strappator...						true	FLOWPACK			
38	RISERVA							FLOWPACK			
--	RISERVA							FLOWPACK			

Table 5.11 Flowpack2 deterministic failures

	string 1	length[mm] 2	speed[m/min] 3	acceleratio 4	real 5	integer 6	boolean 7	string 8	time 9	date 10	datetime 11
string	Name of Attribute	Length	Speed	Acceler...	Real	Integer	Boolean	String	Time	Date	Datetime
17	Pezzi_Stampa					20000		FLOWPACK			
18	T_Cambio_inchiostro							FLOWPACK	10:00.0000		
19	Pezzi_fine_film					40000		FLOWPACK			
20	T_Cambio_film							FLOWPACK	5:00.0000		
21	RISERVA							FLOWPACK			
22	RISERVA							FLOWPACK			
23	RISERVA							FLOWPACK			
24	RISERVA							FLOWPACK			
25	RISERVA							FLOWPACK			
26	RISERVA							FLOWPACK			
27	RISERVA							FLOWPACK			
28	RISERVA							FLOWPACK			
29	RISERVA							FLOWPACK			
30	RISERVA							FLOWPACK			
31	RISERVA							FLOWPACK			
32	RISERVA							FLOWPACK			
33	intervallo_pulizia							FLOWPACK	2:00:00.0000		
34	durata_pulizia							FLOWPACK	1:00:00.0000		
35	RISERVA							FLOWPACK			
36	RISERVA							FLOWPACK			
37	uscita_strappator...						true	FLOWPACK			
38	RISERVA							FLOWPACK			
--	RISERVA							FLOWPACK			

Table 5.12 Flowpack3 deterministic failures

It can be noticed that the two different deterministic failures related to the change of the ink and to the change of the film are set in order to happen in different moments on the different flowpack, considering that:

$$\begin{aligned}
 \text{theoretical flowpacked prod} &= \text{Velocità}_{\text{flowpack}} \left[\frac{\text{pz}}{\text{min}} \right] * 60 [\text{min}] = 600 * 60 \\
 &= 36000 \left[\frac{\text{pz}}{\text{h}} \right]
 \end{aligned}$$

For this reason, the numbers set in the previous tables are all below this value (except the value related to the change of film in the last flowpack, which is slightly higher in order to make the failure occur a bit after one hour of simulation), letting these failures all happen in the course of a single hour.

Moreover, for what concerns the cleaning, it is set to be equal for all the different machines and it should be performed after 2 days, so this kind of deterministic event is not considered in the following test.

The number of emitted products from the source is coherent, since it is given by the following relationship:

$$\begin{aligned}
 \text{modelled prod} &= PR * RM * \text{Produzione minuto} * 60 * \text{Tempo smodellaggio ore} \\
 &= 20 * 4 * 20 * 60 * 2 = 192000 [\text{pz}]
 \end{aligned}$$

In this case, it can be noticed that all the three flowpacks have a comparable number of flowpacked products, since the production number of mobile units per minute “produzione_minuto” is greater than 15, so the third flowpack is not used as jolly in the plant.

In addition, the buffer and the drain contain products for the same reason, since the plant is oversized.

The chart related to the variable “prod_pezziora” referred to any object present in the root frame is in figure 5.10:

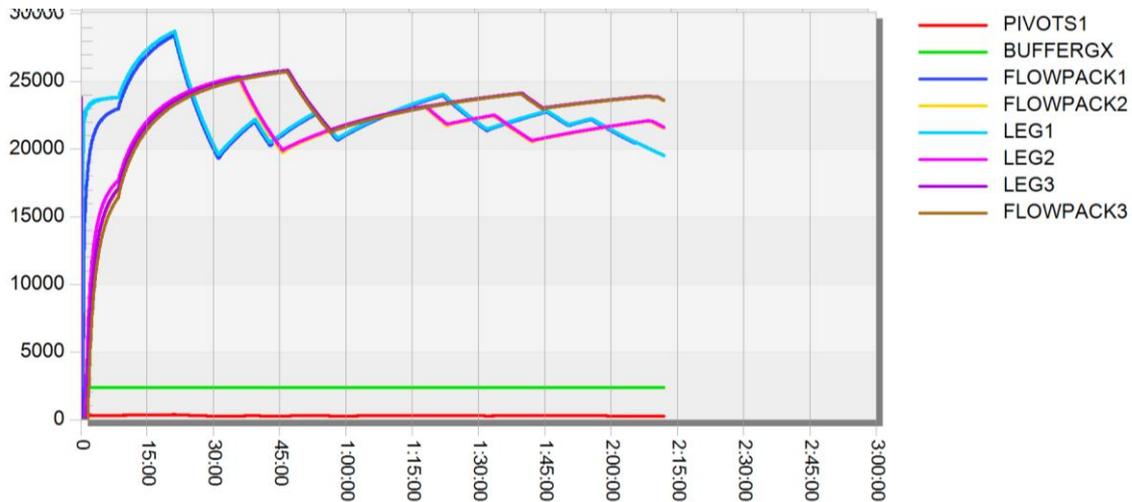


Figure 5.10 prod_pezziora plot

The behaviour of these lines indicates the production per hour and it is coherent with the numbers reported in the displays, since it can be easily seen that the flowpack 1 has a value that lays around 21000 [pz/h] and in fact:

$$prod\ pezziora\ flowp1 = \frac{42960}{2} = 21480 \left[\frac{pz}{h} \right]$$

The same reasonings hold for the other 2 flowpacks. Moreover, it can be noticed that also in this case the lines related to each flowpack are overlapped on the ones related to each leg: this is due to the fact that the products that are flowing in a specific leg are then flowing in the connected flowpack.

For what concerns the graph “Chart_stat”, the obtained histogram is shown in figure 5.11:

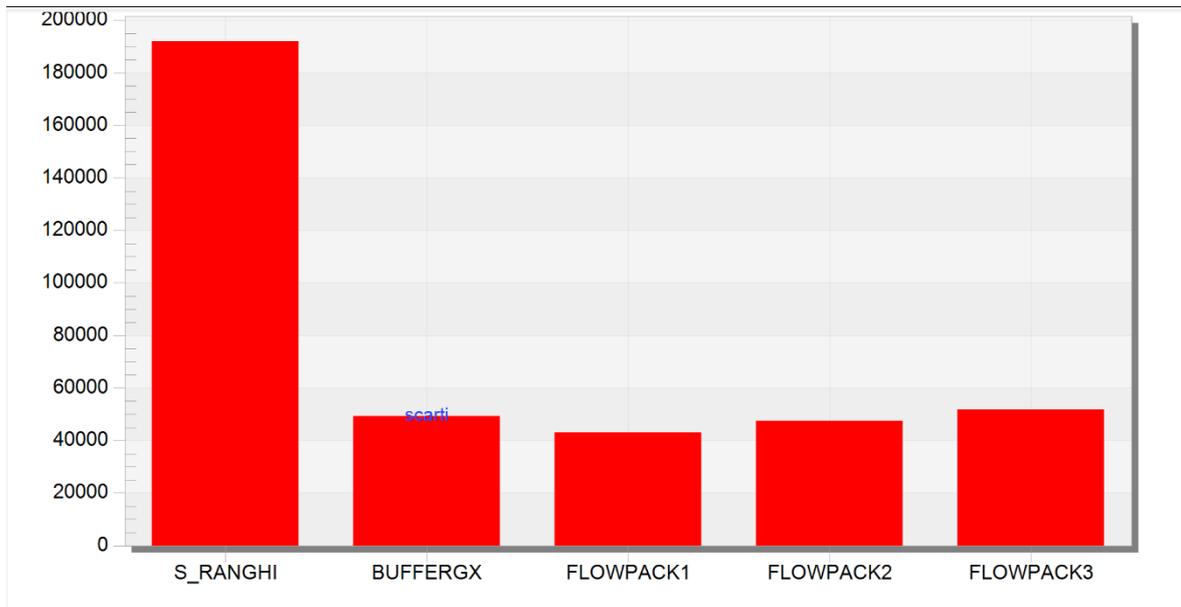


Figure 5.11 Chart_stat histogram

Also in this case, the histogram realistically represents the situation, since the height of the different columns exactly coincide with the values assumed by the displays located in the root frame, either for the flowpacks and for the simple fan buffer.

Finally, the three graphs related to the percentage of usage of the three flowpacks are reported in figures 5.12, 5.13, 5.14:



Figure 5.12 Flowpack1 percentage of usage



Figure 5.13 Flowpack2 percentage of usage



Figure 5.14 Flowpack3 percentage of usage

It is evident that the flowpack 3 is the one that lays more than any other flowpack in waiting, since this wrapper is the last one to be reached by the products. The flowpack 1, instead, is the one with the highest percentage of failure, while the simulation ends in the exact moment when the deterministic failure related to the change of the film, which occurs every 15000 products, is affecting the second flowpack, since its tower has the red colour. Moreover, some products are still present on the leg 2, since the entrance of the flowpack 2 is temporarily locked for the failure occurring.

It must be also underlined that, in this test too, the deterministic events are not plotted as the stochastic failures, but they are considered by means of the final value of the variable shown by the display “PRODOTTI INCARTATI”, because when the events take place, the entrance of the affected flowpack is locked and so the display stops increasing its value till the end of the deterministic event. In addition, the red colour of the tower belonging to a specific flowpack is associated to the deterministic event as well as to a stochastic failure.

In the end, the graphs related to the gondola buffer result the one shown in figure 5.15:

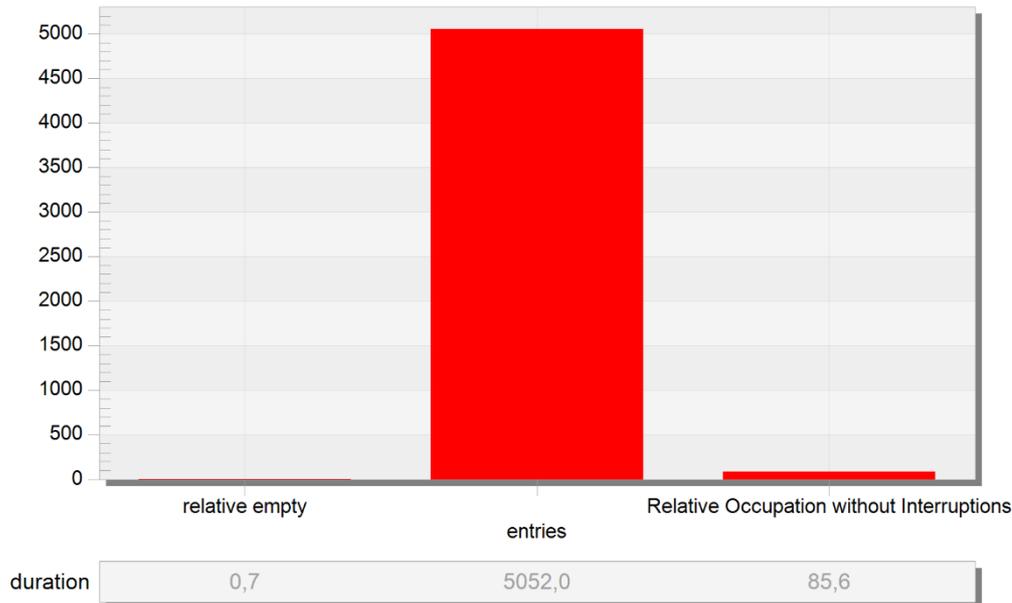


Figure 5.15 Simple fan buffer statistics

This statistics shows that for almost the 0% of the simulation time, the buffer results to be empty and this is coherent with the plant dimensioning, since the line is oversized in order to make the buffer work and store products in itself. The relative occupation with no interruptions is around 85%, showing a high usage of the buffer; the entries are not reported in terms of percentage but in terms of absolute number and they are almost 5050, highlighting again the high work performed by the buffer. Successively, as already explained for the simple fan buffer, also in the gondola buffer the products stored have been recirculated on the flowpack 3 in the first proper moment.

Finally, the efficiency related to the different wrappers and to the whole plant is computed as:

$$\text{Flowpack1 efficiency} = \frac{\text{working time}}{\text{working time} + \text{failed time}} = \frac{56.2}{56.2 + 32.92} = 0.63$$

$$\text{Flowpack2 efficiency} = \frac{\text{working time}}{\text{working time} + \text{failed time}} = \frac{61.5}{61.5 + 20.22} = 0.75$$

$$\text{Flowpack3 efficiency} = \frac{\text{working time}}{\text{working time} + \text{failed time}} = \frac{67.5}{67.5 + 11.36} = 0.86$$

$$\text{Plant efficiency} = \frac{\sum \text{prodotti incartati}}{\text{prodotti smodellati}} = \frac{42960 + 47421 + 52809}{192000} = 0.75$$

For the sake of simplicity, these computations are made by neglecting the products still laying on the line when the simulation stops. Furthermore, it can be noticed that the efficiency is significantly lower than the one obtained in the first test, but this is due to the fact that the line is oversized in order to make some products be discarded in the drain of the buffer and so this lets the efficiency diminish.

Chapter 6 - Conclusions and future developments

Different tests are performed on the two kinds of plants taken into consideration in order to evaluate how they should work in the real environment. Successively, further considerations are performed on the accuracy of the models developed in the virtual environment.

From the analysis performed on the models built in the virtual environment, the expected functioning of the lines is achieved, at least in theoretical and graphical terms. In fact, the lines are suitably set by the data inserted in the frame controls and also the products are suitably sized when emitted from the source. Moreover, the results in terms of modelled products and wrapped products are correct, since the sum of the wrapped products, the eventual buffered products and the discarded ones exactly corresponds to the modelled ones. Moreover, a good level of parametrization of the model has been reached, since no direct mentioning of a specific object in the frames is present, so everything is parametric and suitable for any further and different plant that should be implemented in the virtual environment.

Furthermore, a further step towards the real implementation of these lines has been made by importing also the presence of the different failures that could affect the various machines, also distinguishing them between stochastic and deterministic.

However, a validation of the data and results obtained by means of these simulations should be performed in the real plants, in order to have an estimate of the level of accuracy of the developed models and library.

In addition, further developments tending to the real structure of the different elements can be developed, for instance for what concerns the fan buffer, for which a simple approach has been adopted in order to simplify its structure in the simulator from a graphical point of view, but letting it work in the same way a real fan buffer does.

According to the goal of having test distributions as much as possible similar to the real ones, additional developments of other elements belonging to different plants can be led, such as the building of the BUFFERGLX object, which is a gondola buffer with the products exiting in line and not transversally with respect to the ridge or the REPITCHING object, that is an element whose task consists in equalizing the space laying among the different ranks traveling on the ridge.

Finally, aiming at the maximum level of parametrization that can be reached while developing this library, a reference to the speed related cells of the table of the instances of the models PIVOTSX and PIVOTDX must be added in the reset methods, in order to let the

user just insert the right value in the table DataTable located in frame controls, so automatically assigning the desired speed to the conveyors of the single and double pivot, as done for the other models belonging to the created library.

Bibliography

- [1] J. Kohl, S. Spreng, and J. Franke, *Discrete Event Simulation of Individual Energy Consumption for Product-Varieties*, *Procedia CIRP* **17**, 517 (2014).
- [2] F. Hosseinpour and H. Hajihosseini, *Importance of Simulation in Manufacturing*, 4 (2009).
- [3] O. Berman, *Efficiency and Production Rate of a Transfer Line with Two Machines and a Finite Storage Buffer*, *Eur. J. Oper. Res.* **9**, 295 (1982).
- [4] S. Bangsow, *Tecnomatix Plant Simulation* (Springer International Publishing, Cham, 2016).
- [5] M. Kikolski, *Identification of Production Bottlenecks with the Use of Plant Simulation Software*, *Ekonom. Zarządzanie* **8**, 103 (2016).
- [6] *Plant Simulation Basics*, 518 (2018).
- [7] P. Wegner, *Concepts and Paradigms of Object-Oriented Programming*, *SIGPLAN OOPS Mess* **1**, 7 (1990).
- [8] Simtec, *Dispense Di Simulazione Ad Eventi Discreti Con Plant Simulation*, (2020).
- [9] J. Siderska, *Application of Tecnomatix Plant Simulation for Modeling Production and Logistics Processes*, *Bus. Manag. Educ.* **14**, 64 (2016).
- [10] Ampuero Olga and Vila Natalia, *Consumer Perceptions of Product Packaging*, *J. Consum. Mark.* **23**, 100 (2006).
- [11] M. Semeraro and P. Bo, *INTRODUZIONE ALLE DISTRIBUZIONI RANGHI*, 57 (n.d.).
- [12] M. Semeraro and Paolo Bo, *Gestione Del Ricircolo Di Prodotti Su Una Distribuzione Ranghi in Presenza Di Doppio Carrello Di Ricircolo*, (2013).
- [13] P. B. Domenico Grosso, *Manuale Illustrativo Ed Operativo Delle Gambe Controllate in Gearing*, (2013).
- [14] A. Azzi, D. Battini, A. Persona, and F. Sgarbossa, *Packaging Design: General Framework and Research Agenda*, *Packag. Technol. Sci.* **25**, 435 (2012).
- [15] E. Koenigsberg, *Production Lines and Internal Storage—A Review*, *Manag. Sci.* **5**, 410 (1959).
- [16] *Tecnomatix Plant Simulation Help*, https://docs.plm.automation.siemens.com/content/plant_sim_help/15.1/plant_sim_all_in_one_html/en_US/tecnomatix_plant_simulation_help/tecnomatix_plant_simulation/tecnomatix_plant_simulation_help.html.