Modelling and Simulation of an Automated Warehouse for the Comparison of Storage Strategies

Chapter · February 2010

DOI: 10.5772/7658 · Source: InTech

2 authors:

Valentina Colla
Scuola Superiore Sant'Anna
361 PUBLICATIONS **2,349** CITATIONS
SEE PROFILE

Gianluca Nastasi
Scuola Superiore Sant'Anna
41 PUBLICATIONS **286** CITATIONS
SEE PROFILE

Modelling and Simulation of an Automated Warehouse for the Comparison of Storage Strategies

Valentina Colla and Gianluca Nastasi
Scuola Superiore Sant'Anna
Pisa, Italy

1. Introduction

Among the different kinds of plants that can be simulated, warehouses are surely one of the most common, because of the need of an easy and fast retrieve of the different items to dispatch, independently on the system adopted for their storage. In particular, the progress of industrial automation has promoted the widespread adoption of automated warehouses in a great number of industries of different types. The great advantages that such systems can bring in term of productivity, safety, quality and competitiveness are able to justify their high costs. However the efficiency of any warehouse (automated or not) is determined by factors such as layout, conveyors type, typologies of the products to be stored, shapes of the packages, and storage strategies. Nevertheless, it could happen, after a warehouse had been set up, that some conditions change: production cycles may vary, new product typologies can be introduced, new production lines can be added, etc. so that an intervention on the warehouse in order to keep the desired efficiency becomes necessary. In these circumstances, warehouse layout, product packages and conveyors type are very often constraints, while storage strategies can be relatively easily modifiable. A scrupulous modelling and simulation phase of the warehouse and the eventual automated material handling system could provide a valuable test bench for designing, testing and comparing of new storage polices.

Throughout the chapter a theoretical discussion on the modelling and representation of a warehouse will be provided. An overview of "discrete-event simulation" and its derivate "trace-driven simulation" will be presented: their features will be described as well as two different ways (longitudinal and transversal analyses) to model the problem in order to employ these techniques in an effective manner. A typical automated warehouse will be used as example by characterizing it through events and entities. Besides, the usage of data coming from actual information systems as simulation inputs will be discussed.

Some Key Performance Indicators (KPIs) useful in the evaluation of the efficiency of a warehouse will be defined and used to derive objective functions that need to be optimized in order to improve storage strategies and warehouse performances. As a practical case study, the modelling and the simulation with the trace-driven technique of an existing

automated warehouse for the storage of steel products will be presented. A software system able to simulate the warehouse behaviour will be presented and discussed with the help of the Unified Modelling Language (UML) (Arlow&Neustadt, 2005), which will be used to describe the identified entities, and their relationships. The features of the simulator will be presented as well as its design and some hints on its development in C++.
The modular structure of the software and of its end-user interface will also be analysed in deep details, in order to illustrate how such kind of tools can be used for improving the warehouse management through the development, implementation and test of different storage strategies.
Finally some indications on the storage strategies optimization will be provided and some results will be analyzed by means of the comparison of defined KPIs.

2. The problem

Automated warehouses are complex systems. They often have several constraints due to their layout (single or multi level), the type of material handling system (conveyors, bridge cranes, automated guided vehicle systems, forklift trucks, etc.), the shapes and properties of packages (stacking or not stacking), storage and order picking polices (static or dynamic allocation, first-in first-out, etc.). The optimization of their performances requires the formulation of a model. Previously mentioned elements cannot be easily translated into mathematical terms, thus making difficult the application of traditional operational research techniques (e.g. the simplex algorithm). The risk is in the simplifications that such techniques would require to obtain a usable model (Jones et al., 2002). Moreover, the required optimization is often multi-objective, whose aim is to provide a number of possible solutions that represents different trade-offs between different goals. A better way to cope with this problem consists in describing the model by means of a programming language, that is, in other words, to develop a simulator.
By means of a simulation program it is possible to replicate in a controlled environment the behaviour of a real system in different conditions. More in general, a simulation is an auxiliary tool that allows both analysis and synthesis of dynamic systems, which allows to measure the performances of existing systems for different structural features as well as to assess at design-time the behaviours of the system for different operative conditions.
The importance of simulation is highlighted by the following tasks it can accomplish:

- Study and analysis of interactions between single components of complex systems;
- Impact assessment of potential changes to an existing system;
- Design-time evaluation on the performances of a system for different operative conditions;
- Performance analysis of an existing system;
- Generation of an artificial history of a system;
- Parameters optimization;
- Analysis of possible bottlenecks;
- Capacity planning;
- Assessment of performances of different systems in the same conditions;
- Analysis of system behaviour in rare and risky conditions;
- Estimation of not-measurable variables and quantities;
- Exploration and assessment on new management policies.

In this optic, a general purpose programming language (e.g. C/C++, Java) allows describing the model with the necessary degree of detail. Some tools already exist that can make easier the duty of building a simulator, such as GUI-based software and framework libraries. Nonetheless they have some limitations, as will be explained in the next section, that do not allow a complete customization and an exploitation of most advanced optimization techniques.

3. Approaches to the simulation of a warehouse: existing tools

Warehouse design or re-design often require the use of a simulator for different reasons:

- provide a proof of concept;
- test warehouses performances with different throughput;
- optimize the layout;
- optimize storage strategies;
- answer "what if" questions.

The problem is common and, over the years, different tools have been developed in order to help designers and users. Existing tools can be divided in three main categories: GUI-based simulation software, framework libraries and specialized programming languages. In the following sections these solutions will be briefly illustrated and some available tool will be cited.

3.1 GUI-based simulation software

This kind of software is internally very complex and they have very high costs. They allow building simulators visually, relatively quickly and with a minimal effort by using graphical elements and libraries of entities, without the need to write code. Most of these tools consist of two components: a build environment, which allows the definition of the physical and the logical model of the plant by means of component libraries and CAD tools, and an engine, which interprets these models and actually simulates them. Advanced tools (e.g. AutoMod™ or Siemens Tecnomatix™) can also visualize the simulation in a 3D environment (LeBaron&Jacobsen, 2007).
One of the key features of the GUI-based simulation software that makes them so quick and easy to use, i.e. the component library, is also one of their limitations. Actually not all warehouses employ standard elements and the implementation of new modules in these environments are not straightforward.
Besides, the investments necessary to buy this kind of software can be hardly justified when there is the need to simulate only one plant. In facts, the targets of these systems are more likely to be warehouse designers and builders or very large companies.
Another limitation is represented by the difficult integration of custom advanced optimization techniques, like those that will be presented after in this chapter.

3.2 Framework libraries

The aim of this kind of solutions is to help programmers developing custom simulators by providing a set of standard objects, methods and functions, which may be used to build a simulation engine. There exist several libraries for almost all known general purpose

programming languages (especially Java), both freeware and commercial, and most of them are based on “Discrete-Event Simulation” (DES) principles.
Unfortunately, the majority of these tools is oriented to networks simulation (e.g. OMNeT++ and Ns2) (Varga, 2001) and cannot be adapted to the simulation of warehouses in a simple way (if the license allows it). Moreover, with the existing framework libraries it is difficult to exploit historical data obtained from warehouses information systems in order to drive the simulation by means of known inputs, therefore the development of a “Trace-Driven Simulation” (TDS) is not possible or it requires too much effort.

3.3 Specialized programming languages

First specialized programming languages oriented to simulation were developed during '50s and '60s in the context of manufacturing companies in order to speed up the realization of simulation models, which were, until that time, made in Fortran, Assembler or even manually.
Most popular and used languages were GPSS (Gordon 1962) and SIMSCRIPT (Markowitz 1963) and SIMULA (Dahl&Nygaard 1966). These languages and their evolutions are nowadays encapsulated in advanced build environment, like those presented in Sec. 3.1.
If on the one side the specialization of this kind of languages helps by simplifying the modelling of the problem thank to a targeted set of constructs and instructions, on the other side they impose some restrictions and boundaries, as in the case of framework libraries.

4. Warehouse modelling

As highlighted in previous sections, using existing tools could be of great help, although in some cases the development of a custom simulator becomes necessary.
In literature three main simulation paradigms do exist (Borshchev&Filippov, 2004): System Dynamics (SD), Discrete-Event Simulation (DES), and Agent Based (AB). The first one is the less recent and it is based on the concepts of stocks (e.g. of material, money) and continuous flows between them; it aggregates single entities and events, concentrating on polices and strategies. DES is a well-known and widely adopted paradigm, also by commercial software. In DES entities are passive and respond to predetermined events, which are organized in an ordered queue. AB is, vice versa, a relatively new paradigm, used mainly in academic researches, mainly suitable for social and biological studies; it is based on active and autonomous entities that define in a bottom-up way the global system behaviour by means of their interactions.
The choice of which model to adopt in the simulation of an automated warehouse naturally falls on DES: in fact such a system is actually a collection of entities (e.g. production lines, conveyors, cranes) that respond to fixed events (e.g. production orders, picking messages).
In this section an overview of DES and its derivates will be presented as well as two different ways (longitudinal and transversal analyses) to model the problem in order to apply these techniques in an effective manner.

4.1 Discrete-Event Simulation

To better understand what DES is, let's introduce some definitions (Banks et al., 2004):

- *clock*: variable representing the simulation time;

- *system state*: collection of variables that contains all necessary information needed to describe the system itself, especially those that shall be analyzed;
- *event*: instantaneous happening that changes the system state; an event is defined as *conditioned* or *dependant* if it always occurs together with another event, *primary* or *independent* vice versa;
- *pending events queue*: ordered list of events for which the occurrence time is known;
- *entity*: system component that requires an explicit representation of its own model;
- *attributes*: properties that describe a certain entity;
- *entity set*: temporary or permanent collection of entities organized by a certain logic;
- *activity*: time interval of specified amount during which determined actions are carried out by a single entity or by an entity set;
- *delay*: time interval of unspecified duration.

In DES the system evolves according to the program shown in Fig. 1. At the beginning the system, the events queue and the clock are initialized: an initial state for the system and all its entities is assigned, the queue is filled with programmed events and the clock is reset. Subsequently, as long as the events queue is not empty, the simulation enters in its main loop where the nearest event is scheduled by updating the clock to the event time. The event is than signalled to the system that propagates it to all its entities. Entities affected by the event carry out defined actions, which, in turn, have two side effects: the system state changes and new events could be triggered and pushed into the events queue. When there are no more events, or a stop condition is met, the simulation breaks the loop and a report of the observed system state variables should be returned in order to be analyzed.

```
system.init();
events_queue.init();
clock.reset();
while( !events_queue.empty() )
{
   event = events_queue.next();
   clock = event.time();
   system.signal(event);
   system.update_status();
   events_queue.update();
}
system.report();
```

Fig. 1. Discrete-Event Simulation main loop

4.2 Model characterization

The most critical phase in DES modelling is, without any doubt, the identification of entities, activities and events that characterize the system and its evolution.

Two methods will be here reviewed: *longitudinal* (or *by process*) and *transversal* analyses. Both methods are based on the distinction between *resident* (e.g. production lines) and *transient* (e.g. semi-worked products) entities (Bratley et al., 1987).

Adopting the first form of analysis means representing the system dynamics - i.e. the sequence of events - as flows of transient entities. For example, let us consider a production system in which material is handled from one equipment to another along a certain transformation process. Thus, using longitudinal analyses, semi-worked items correspond to transient entities, which are affected by a sequence (a list of events) of transformations

(activities) carried out by various equipments, i.e. resident entities are here seen as resources (of work). At the end of each activity, the attributes of transient entities establish the system state and they can trigger new events (e.g. alarms).
Vice versa, the adoption of transversal analysis implies the description of the system dynamics in term of cycles of resident entities. Let us apply this method to our example: each machine carries out a cyclic sequence (events list) of operations (activities), during which a new resource is transformed in a semi-worked product. At the end of each cycle, attributes of resident entities establish future events and they modify the state of transient entities.
Nevertheless, it is always convenient to apply both analyses in order to spot every event or entity and then to simplify the model by eliminating superfluous and redundant elements.

4.3 Simulation initialization

In order to start a simulation, an initialized pending events queue is necessary. The initialization can be done in various ways:

- *random initialization* can be used when no prior knowledge of events timing is assumed;
- *stochastic initialization,* on the contrary, can be employed when an *a priori* characterization of events statistics is provided (e.g. frequency, distribution);
- *trace initialization* is applied when historical or log data regarding events can be obtained from, for example, an information system (e.g. production orders, malfunctions).

When the last type of initialization is used, the simulation takes the name of *Trace Driven Simulation*, which is a well-known technique especially in the field of computer architecture evaluation (e.g. cache, memory) (Fu&Patel, 1994), but that can easily be extended to other traceable systems, like warehouses, for example.
On the one hand, this kind of simulation provides some key advantages by using "real" data:

- simulation credibility;
- easy and straightforward model validation;
- direct and impartial comparison between simulated results and measurements on the real system.

On the other hand, this method can be applied only when the system to simulate, or a similar system, already exists and data logs are available.

5. Key Performance Indicators

In order to assess, compare and improve the performances of a warehouse and employed storage strategies, it is convenient to define a set of Key Performance Indicators (KPIs) (Chan, 2003). Such KPIs can be used as system state variables or even as objective functions of an optimization system linked to the simulator.
Generally KPIs should be customized on the particular warehouse system in exam so as to adapt them to its peculiarities and to emphasize any desired aspects.

In this section some general KPIs will be presented together with some more specialized indicators, suitable for warehouses organized in aisles with dynamic compartment allocation and products stored in stacks (see Sec. 7).

5.1 Throughput

Throughput the measure of the number of items that enter or exit during a time unit (hour, shift, day)

$$T_{in} = I_{in} / \Delta t \tag{1}$$

$$T_{out} = I_{out} / \Delta t \tag{2}$$

where T is the throughput, I is the number of inbound (in) or outbound (out) items.

5.2 Average Stock

It expresses the average quantity of products stored in the warehouse during a certain amount of time.

$$\overline{S} = \frac{\sum_{i=1}^{N} S(i)}{N} \tag{3}$$

where N is the number of observed days, S(i) is the stock measured the i-th day, which can be expressed as number of products or total weight (it depends on the typology of material).

5.3 Receptivity

It is a measure of the quantity of items a warehouse can host. For warehouses that use static allocation, receptivity is rather simple to calculate; otherwise it's assumed to be approximately equal to the maximum quantity of items that has been recorded or during the simulation or by the information system.

5.4 Receptivity Saturation Coefficient

It express how much the warehouse has been exploited in a certain period of time

$$RSC = \overline{S} / R \tag{4}$$

where $\overline{S}$ is the average stock in the observed period and R is receptivity.

5.5 Handling potentiality

In Automatic Storage and Retrieval Systems (AS/RSs) it measures the average number of handled items as

$$HP = \frac{\sum_{i=1}^{N} I_{in}(i) + I_{out}(i)}{N} \tag{5}$$

where I_{in} and I_{out} are, namely, the number of inbound and outbound items in the i-th time interval and N is the total number of considered time intervals.

5.6 Fragmentation

It is a measure of space breakup, useful for warehouses with dynamic compartments allocation. It is defined as

$$F(i) = 1 - \frac{NS_0}{NS(i)} \tag{6}$$

where NS_0 is the number of empty spaces present at time t=0 (e.g. in a warehouse organized in n aisles $NS_0=n$) and NS(i) is the number of empty spaces in the i-th observed time interval.

6. Programming tips

In this section some hints regarding the implementation of a warehouse simulator will be illustrated, with particular emphasis on object-oriented programming, reusable design patterns, data structures, and maintenance. The reference programming language is C++, but the concept hereafter exposed can be simply extended to other high level programming languages.

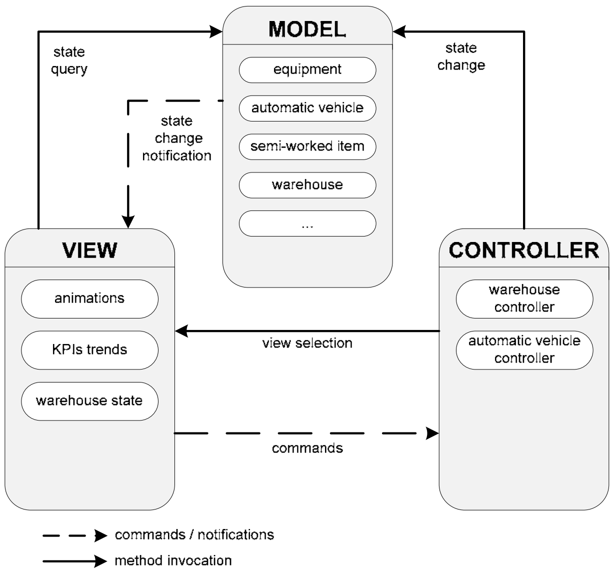

Fig. 2. Example of Model-View-Controller design pattern

6.1 Model-View-Controller

In interactive simulations, the Graphic User Interface (GUI) offers a direct visualization of the dynamic behavior of the modelled system and it provides to the analyst a way to directly interact with the simulation that he is executing. Typically the software modules dedicated to visualization and user-interaction are integrated in the module responsible of the model simulation. Nevertheless such integration makes hard the design and especially the maintenance of complex simulators and it imposes unnecessary constraints to GUIs development (Krasner&Pope, 1988). In order to obtain more robust and easier to maintain simulators, the software should be designed following the *Model-View-Controller* (MVC) design pattern: *models* are represented by identified entities, *controllers* implement models logic and are responsible for state transitions, while *views* manage GUIs and user interactions by routing commands to *controllers* (Fig. 2). The MVC design pattern has been employed with success in several complex simulation applications (Narayanan et al., 1997). The net separation of duties simplifies modeling and programming, it makes the system more robust and it eases debugging and maintenance of the application.
Models, views and controllers should be mapped on separated classes: even if this technique probably produces a higher number of classes, some of them are very simple, especially models, so that however the complexity decreases and the code become more linear.

6.2 Strategy design pattern

Strategy is a behavioral design pattern (Gamma et al., 1994) that allows a high degree of modularity in the implementation of different strategies and polices (e.g. storage, picking, ordering strategies) by delineating a family of algorithms through the definition of a common interface and by making them easily interchangeable. By means of this design pattern it is possible to choose which polices to employ at run-time and, besides, it makes very easy the addition of new strategies, also after the software is already developed. In Fig. 3 the UML diagram of the classes is shown: classes that implements concrete strategies (*ConcreteStrategyA* and *ConcreteStrategyB*) are derived from the abstract base class *AbstractStrategy*, which declares an abstract method *algorithm()*. The *Context* class has a pointer to the abstract base class, to which the address of a particular instance of one of the concrete derived classes is assigned on the base of the strategy that has to be applied.

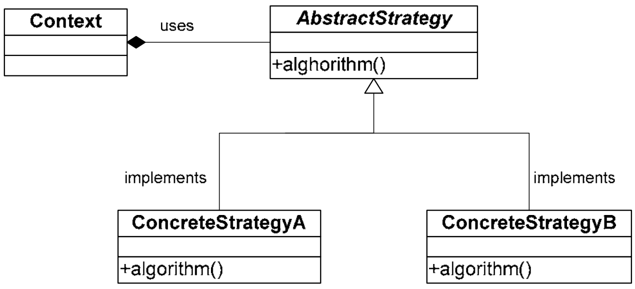

Fig. 3. UML class diagram of a typical Strategy pattern.

For example, let us consider a warehouse with three different allocation strategies (StrategyA, StrategyB and StrategyC), where each of these polices tends to optimize a

different KPI. The inputs for the three algorithms are the same: the item to be placed and the state of the warehouse. A common interface can therefore be defined by means of an abstract class *AllocationStrategy* that declares an abstract virtual method *allocate(item i, warehouse w)*. This method is therefore mandatorily implemented by subclasses, which will encapsulate the different strategies.
When, for example, it is necessary to allocate an item by means of *StrategyA*, an object of this type is instantiated and its address assigned to the pointer owned by *WharehouseController*. When it will call the *allocate* method, the invocation will be translated, thank to the polymorphism rules, in an invocation of the allocation method of the concrete class *StrategyA*.
By applying this scheme it is therefore possible to realize different families of strategies (allocation, reordering, picking) each one containing several possible strategies able to optimize different KPIs or that employ different optimization techniques.

6.3 Visualization and animations

Sometimes it could be convenient to visualize interactively the evolution of the simulation: charts showing trends of main KPIs, diagrams representing allocated compartments and animations of the material handling system could be very helpful for both the researcher and the final user.
While charts are relatively simple to implement thank to the great number of both freely and commercially available modules, custom animations or diagrams could be more tricky to develop. GDI+, DirectX and OpenGL are only few of the technologies that can be used to realize such elements: the first one is helpful for bi-dimensional diagrams and animation, while the other two are employed mainly for coding 3D environments.
Although all these technologies are well documented, the design of data structures that models such animation remains the most difficult step to accomplish. Unfortunately there are not general data structures that can be employed out-of-the-box, but they should be designed and customized around the particular problem.
However, for the sake of example, let us consider a material handling system composed by automated vehicles linked to rail tracks. Each vehicle is autonomous and it is able to respond to mission messages (e.g. pick an item, store an item). Such system could be modelled by means of a graph: rail tracks can be represented by weighted arches, where weights are proportional to trunks length and nodes represent tracks intersections. Vehicles are autonomous entities, thus coded by means of instances of vehicle classes. The position of a vehicle is determined by the arch to which it is linked and by the amount of covered space on that arch. At each clock event, vehicles move forward by a certain amount of space, proportional to their speed. Each vehicle instance checks if the track portion it has to cover is free in order to prevent collision and railroad switches are driven so as to respect precedence and priorities.

7. A case study

In this section a warehouse simulation case study will be presented. The actual plant will be illustrated as well as the storage strategies optimization problem that makes necessary the development of a simulator. A description of the simulation software will be then provided by means of UML diagrams and screenshots.

7.1 The plant

The analyzed plant (Fig. 4) produces steel tubes of different qualities, lengths and shapes, which are stored and sold in packs of the same typology. The plant is composed by an area where tubes are manufactured by means of four production lines (profilers) which are able to lengthwise solder steel strips and to produce tubes with different shapes. These lines are connected by means of an automated material handling system to a storage area where final products are stored until customers order them: this system is composed by 43 peculiar automated cranes, called trolleys, which can move by means of a rail tracks suspended to the plant ceiling. The trolleys are equipped with electromagnets capable to capture one pack of tubes, they are capable of self-governing movements thanks to an on-board Programmable Logic Controller (PLC) and they can also communicate with each other and with the warehouse main PLC by means of radio-frequency transceivers. The task of the main PLC is to assign missions to trolleys which can be of three types:

- pack retrieval at the end of one of the production line, transportation to the storage area and subsequent stacking;
- pack retrieval in the storage area for dispatching;
- pack reordering inside the storage area.

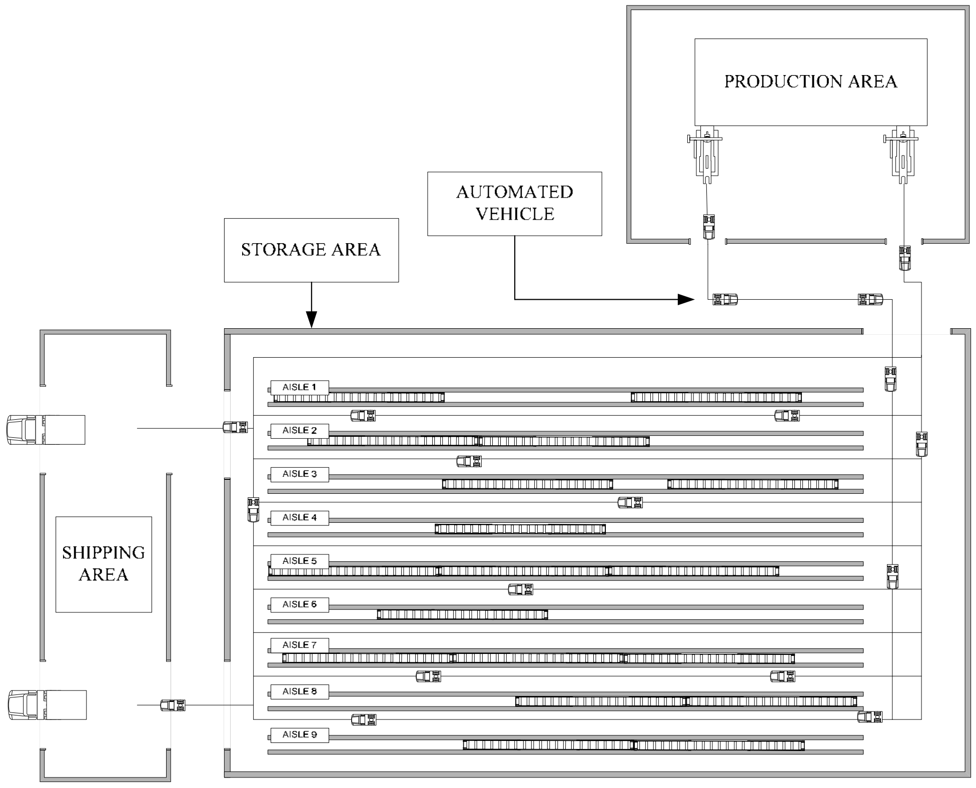

Fig. 4. Plant layout.

The warehouse is connected to the information system of the company, thanks to which it receives production and dispatching orders, by thus minimizing the human intervention. The storage area is divided in 9 aisles that are 90 meters in length, where tubes are stored in dynamically allocated compartments, each one composed by the same typology of product (there can be more than 1000 different typologies). Each compartment is in turn composed by one or more stacks of packs, where the number of stacks is calculated in order to ensure a steady structure and is a function of the number and section dimensions of packs to store (each typology has default pack sizes). Compartments take up the entire width of an aisle and are placed with a predetermined slope with respect to the aisle itself due to the motor position on the trolleys and to the path of the rails. Furthermore, three big pillars (called physical poles) are placed in each aisle to prevent accidental falls of all the stacks.

7.2 Storage strategies

As previously described, the allocation subsystem is entirely automated: when a production order is received by the central mainframe, the latter sends an allocation order to the warehouse management system, which in turn pre-allocates and configures an adequate compartment. If a compartment suitable for a peculiar typology of tubes is already available, the system exploits it, otherwise a new compartment is allocated: the stack composition and compartment occupation are calculated and a suitable free space is picked out. The algorithm looks for available empty spaces in the aisles, which are coded by their starting position and width, and chooses the first empty space that allows to insert a new compartment. If it is not possible to insert that stacks configuration, a new stacks configuration will be calculated, by decrementing the compartment by one stack at time. This subsystem is based on heuristic rules which tend to optimize free linear space and favour distribution of packs along different aisles in order to have a sort of redundancy that enhances insertion and drawing parallelism, by also limiting conveyors traffic problems. In the reordering subsystem, up to now three different strategies are implemented:

- *FIFO reordering*: In order to prevent tubes oxidation, formerly produced packs that belong to almost empty compartments are put over new ones. Handlings for FIFO reordering are carried out by considering:
  - the number of packs in the considered compartment;
  - the compartment ageing;
  - the number of activation of the FIFO reordering in a day.
- *Stacks reordering*: this kind of reordering consists in compacting compartments containing few packs, so that they can be rearranged in a compartment with fewer and higher stacks. Stacks reordering are carried out by considering:
  - the maximum number of packs to reorder;
  - the maximum number of activations of the stacks reordering per day.

  When there are some stacks which satisfy these two conditions, a new compartment is created with a more compact configuration.
- *Space reordering*: the goal is to defragment free spaces in order to obtain bigger free spaces instead of many small ones. This strategy is carried out by considering:
  - the number of movable packs;
  - the size of adjacent empty spaces.

7.3 The problem

The illustrated strategies actually employed in this plant are based on heuristics and do not allow a full exploitation of the available storage space so that it became necessary a precise study with the aim of improving warehouse performances. Simulation is a basic step toward performances optimization of the automated warehouse because direct trials on the real system are obviously not possible and a simulator allows a comparison of the results of different storage strategies in terms of KPIs values and trends.
Although some general warehouse simulation tools already exist, as depicted in Sec. 3, the peculiarity of the considered system and the will to employ advanced optimization techniques imposed the design of a specific simulator.

7.4 Entities and events identification

By applying both analysis methods presented in Sec 4.2, the model can be characterized by means of the identification of entities, events and activities. As the longitudinal analysis is concerned, tube packs can be identified as the transient entities, characterized by quality, length, and shape (properties) and by activities like production, handling, storage and dispatching. Each activity begins with an event generated or by the information system (e.g. production order, dispatching) or by the end of a previous activity (e.g. the handling of a pack starts after the end of its production). Vice versa, by using the transversal analysis, three different types of resident entities can be identified in profilers, automated cranes and warehouse management system. Each of these entities is in fact characterized by cycles of activities - the same listed before -, which in turn operate on the transient entities (tube packs). The initialization of the event queue is done on the basis of data extracted from the plant information system (trace driven simulation - see Sec. 4.3), which contain the details about production and dispatching orders (date, time, typology).

7.5 Design and implementation

In the design of the simulator the techniques presented in Sec. 6 are employed. In particular, the MVC design pattern fits very well with DES entities decomposition. Each entity in facts can be mapped on a model, i.e. on a class in which members represent properties and internal state of the entity and methods represent activities. Some controllers should be also defined: a warehouse controller, responsible of the management of storage space which implements storage strategies and logics, a vehicle controller, responsible of managing vehicles missions and rail switches and a simulation engine, which owns the pending events queue and the simulation clock. Finally, several types of views can be added as needed, allowing an interactive visualization of the simulation evolution: for example a view that summarize the plant state (orders in production, quantity of stored products, allocation state) as illustrated by the screenshot in Fig. 5, a view showing KPIs trends by means of charts or a view showing the automated vehicles system animation. In Fig. 6 an UML diagram of the classes that compose the simulator is shown.

7.6 Strategies optimization and comparison

As described above, the main aim of the simulator is to provide a benchmark through which optimization, testing and comparison of new storage strategies can be done. As a first step towards the optimization, the current storage strategies were reviewed and modified in order to be more rigorous by including strict objective functions. This work, discussed in (Colla et al., 2008), allowed a clean improvement of all KPIs, by providing an increment of the total amount of storable products.

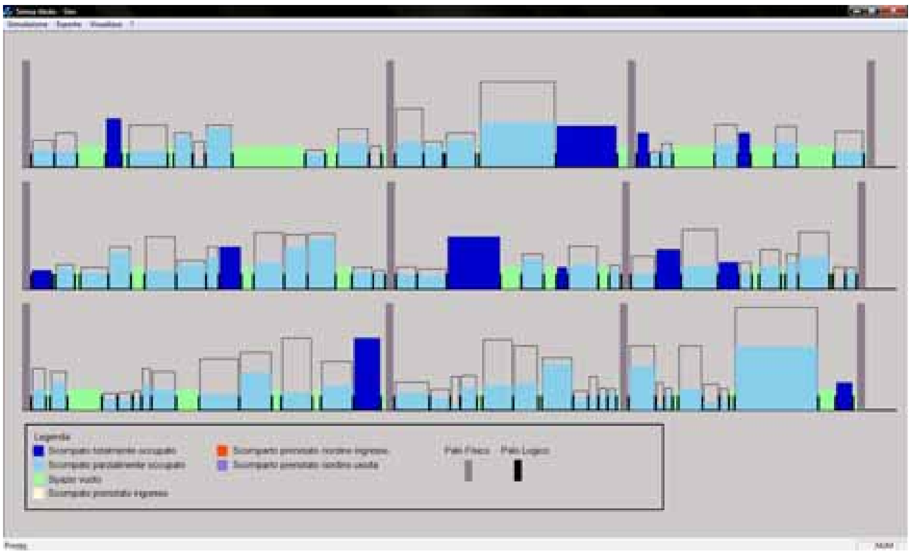
Legenda
Scomparto totalmente occupato
Scomparto parzialmente occupato
Spazio vuoto
Scomparto prenotato ingresso
Scomparto prenotato riordino ingresso
Scomparto prenotato riordino uscita
Palo Fisico
Palo Logico

Fig. 5. Screenshot representing the allocation state of three aisle

More advanced techniques based on genetic algorithm are still under development.
The results of each simulation can be then exported as spreadsheets containing KPIs values sampled at a predefined frequency in order to allow easy comparisons, statistic evaluations and data analyses.

8. Conclusion

A typical warehouse simulation problem has been presented in Sec 2. In Sec. 3 existing free and commercially available tools for warehouse simulation have been described, presenting pros and cons of their adoption. Sec. 4 is devoted to the illustration of Discrete-Event Simulation and its derivates, providing modelling and analysis methods. In Sec. 5 typical Key Performance Indicators are derived. Some hints and tips are provided in Sec. 6, which may help in the development and programming of simulator software. Finally, in Sec. 7 a case study of the implementation of a simulator of a real warehouse for storage strategies comparison is described.

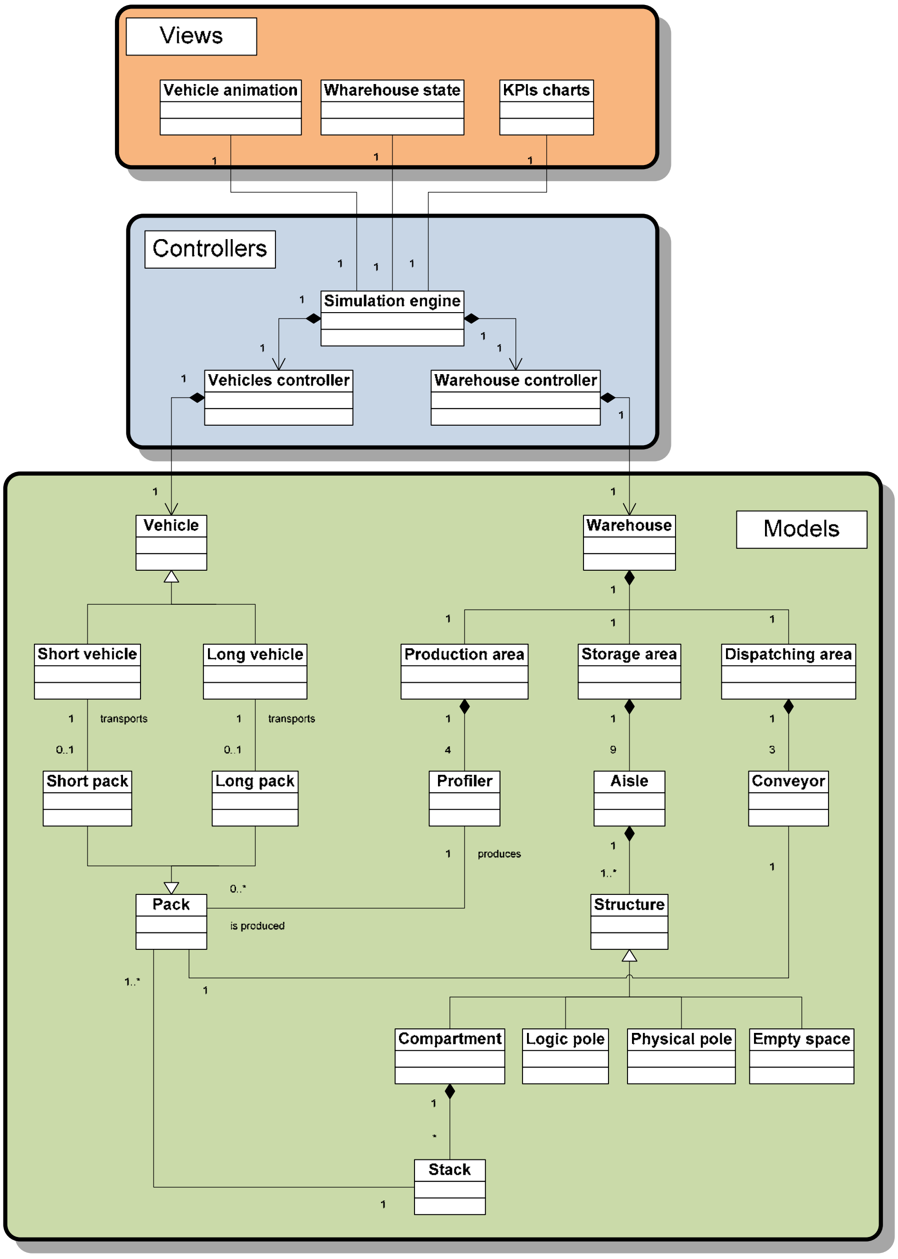

Fig. 6. UML diagram of the classes illustrating MVC design pattern decomposition.

9. References

Arlow, J. & Neustadt, I. (2005). *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley, ISBN 978-0321321275

Banks, J.; Carson, J.; Nelson, B.L. & Nicol, D. (2004). *Discrete-Event Simulation*, Prentice Hall, ISBN 978-0131446793

Borshchev, A. & Filippov, A. (2004). From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools. *Proceedings of the 22nd International Conference of the System Dynamics Society*, July 2004, Oxford, England

Bratley, P; Fox, B.L. & Schrage, L.E. (1987). *A guide to simulation*, Springer, ISBN 978-0387964676

Chan, F.T.S. (2003). Performance measurement in a supply chain. *The International Journal of Advanced Manufacturing Technology*, Vol. 21, No. 7, pp. 534–548, ISSN 1433-3015

Colla, V.; Nastasi, G.; Matarese, N. & Ucci, A. (2008) Simulation of an automated warehouse for steel tubes, *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pp.150-155, ISBN 0-7695-3114-8

Dahl, O. & Nygaard, K. (1966). SIMULA, an Algol based simulation language. *Communications of the ACM*, Vol. 9, No. 9, 1966, pp. 671-678, ISSN 0001-0782

Fu, J.W.C. & Patel, J.H. (1994). Trace driven simulation using sampled traces, *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, Vol. 1, pp 211-220, ISBN 0-8186-5090-7, Wailea, HI, USA

Gamma E.; Helm, R.; Johnson, R. & Vlissides, J.M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, ISBN 978-0201633610

Gordon, G. (1962). A general purpose system simulator (GPSS). *IBM Systems Journal*, Vol. 1, No. 1, pp. 18-32, 1962

Jones, D.F.; Mirrazavi, S.K. & Tamiz, M. (2002). Multi-objective meta-heuristics: an overview of the current state-of-the-art. *European Journal of Operational Research*, Vol. 137, No. 1, (Feb. 2002), pp. 1-9, ISSN 0377-2217

Krasner, G.E. & Pope S.T. (1988). A Cookbook for Using the Model-View-Controller User-Interface Paradigm in Smalltalk-80, *Journal of Object-Oriented Programming*, Vol. 1, No. 3, September 1988, pp. 26-49, ISSN 0896-8438

LeBaron, T. & Jacobsen, C. (2007). The simulation power of Automod, *Proceeding of Simulation Conference, 2007 Winter*, pp. 210-218, ISBN 978-1-4244-1306-5, Washington DC, Dec. 2007

Markowitz, H.; Karr, H.W. & Hausner, B. (1963). *SIMSCRIPT: a simulation programming language*. Prentice Hall, Englewood Cliffs

Narayanan, S.; Schneider, N.L.; Patel, C; Carrico T.M.; DiPasquale, J. & Reddy, N. (1997). An object-based architecture for developing interactive simulations using java. *SIMULATION*, Vol. 69, No. 3, pp. 153-171, ISSN 0037-5497

Varga, A. (2001). The omnet++ discrete event simulation system. *Proceedings of the European Simulation Multiconference*, pp. 319- 324, Prague, Czech Republic, June 2001, SCS - European Publishing House