# A standard approach to production systems modelling based on Finite State Automata

Giacomo Tavola, Marco Taisch, Filippo Boschi
Department of Management, Economics and Industrial Engineering
Politecnico di Milano
Milano, Italy
[giacomo.tavola marco.taisch, filippo.boschi]@polimi.it

*Abstract*— **Simulation has been used for decades as a tool to support decision making in manufacturing systems. It is far cheaper and faster to build a virtual system and experiment with different scenarios and decisions before actually implementing the system. Simulation has been widely used to support decisions in manufacturing systems' operations and configuration. Nevertheless, knowing the changeable nature of manufacturing systems that is affecting the current production environment due to smart technologies adoption and market demands, a new approach to model a typical production system based on a modular and lean paradigm is needed to make the simulation methodologies aligned with "fast reconfiguration" paradigm. In this context, the objective of this paper is to propose a novel approach to create a production model leading to speed-up the simulation of the dynamic behaviour of a flexible and reconfigurable production system. In the first part of this paper, a modular representation of a generic production line is proposed based on a reduced standard set of components. In particular, independent standard entities representing each actor involved within a typical production line (i.e. machines, buffers, etc) are introduced. Moreover, presenting the rules that regulate how these entities can interoperate each other, and a modular, scalable and interoperable model to develop an adaptive, changeable simulation activity is shown. Finally, the instantiation of specific configuration based on different modular entities, using Finite State Automata (FSA) formalism and its application on Industrial use case is depicted; implemented configurations can be executed on different technology platforms (MS-Excel/VBA, Matlab-Symulink or others).**

***Keywords—Production Systems, simulation, modelling, Finite State Automata***

## I. INTRODUCTION

Simulation and optimization of Discrete Event Systems (DESs) can be performed using two different modules. The first one is based on simulation activity and it aims at evaluating the system performance while the second one carries out the optimization techniques for the generation of the optimal configuration [1]. As far as the simulation techniques are concerned, earlier researchers have recognised the problems in representing the manufacturing system especially in how they are controlled [2],[3]. Earlier tools were more complex with limitations in terms of flexibility in representing reconfigurable systems [4].
In fact, these simulation activities adopted for production process analysis are usually based on predefined models. In this context, the behaviour of any production system, controlled, for instance, with a Kanban logic or with a typically push logic (i.e. Base-Stock Control Systems) is usually developed in previous studies by means of already existing standardized models [5]. Hence, having already determined a priori the logic and the operation of production control system, such models provide immediately all production parameters, which are subject of the simulation itself. This problem arises when the process to be analysed is not exactly replicable with one of such models. This case demonstrates the need to use elementary, modular and interoperable entities allowing to compose a typical production process in order to model any process, from the more common to the more complex.
Therefore, this concept leads to readapt the approach to be adopted before simulation activity changing the question of 'which model is most appropriate for replicating a specific production logic?' to 'what combination of standard components is the most appropriate?'

For this reason, new approaches for modelling manufacturing systems are needed to face the emergence of object oriented design approach and so also modular design and software reusability topics gained increased interests [4].

The initial manufacturing system model must capture the nature of the system by describing system objects, flows, systems' functions and processes. Hence, following the literature review, a modelling language for system design must be able to describe system entities, functions and behaviour and it should provide a unified platform that integrates them [6]. Moreover, it should be scalable, changeable and interoperable. Finally, it should be simple and understandable to developers and system experts and capable of representing the system in a common software [4];

Following these characteristics, the objective of this paper is to describe an approach for the design of discrete event simulation (DES) models of large reconfigurable and flexible manufacturing systems based on a limited set of finite state automata (FSA) implementation for modular manufacturing entities and relationships. The proposed approach implies a Discrete Event Simulation (DES) as it is based on a physical system representation based on mathematical/logic model able to point out state changes at precise points in simulated time

[7]. Moreover both the nature of the state change and the time at which the change occurs mandate precise description. These mathematical/logical architectures are included in the Finite State Automata (FSA). For this reason, the proposed approach refers, also, to Agent-based (AB) simulation as its specific instance (Finite State Automata) are, in accordance with Wooldridge and Jennings, autonomous, proactive, reactive and "social" as they can communicate or interact with other systems [8].
Moreover, this approach simplifies the process of creating new manufacturing scenarios where it can be frequently modified without any depth knowledge of simulation language by essential shop floor system parameters and logics and simplified description of relative interconnections.
Chapter II presents the set of elementary entities beneath the proposed approach considering the way single production entities (mainly machines and buffers, but also operators and applications) behave based on the interaction they have with surrounding entities, which means the way they exchange signals to coordinate material flows, and the basic working parameters as duration of operations and setups for different product types.
Chapter III describes how it is possible the implementation of the modelling of a production system utilizing the basic components and eventually come to a complex system. Some specific functional modules are described as well to demonstrate how complex workflow can be easily represented.
Chapter IV presents how the basic functional block are implemented based on Finite States Automata (FSA) formalism. Each implemented automata is described on a simple sheet utilizing a limited instruction set and, if needed, can be amended.
The Chapter V depicts the simulation of a system both from the product perspective (single products, lots and associated parameters) and the resource perspective (availability of machine, faults, setup, repair).

## II. Modelling a Production System with Standard Components

A typical production system can be described as a collection of interconnected equipment (i.e. workstations and buffers) and human resources, whose function is to perform one or more processing and/or assembly operations on a starting raw material, part, or set of parts [9].
In such system, a discrete flow of material can be regulated with different production logics that can be distinguished in a pull or push logic control system [10],[11].
The characterizations of the push/pull distinction that have appeared in the literature can be summarized into the following three definitions:

1. A pull system initiates production as a reaction to current demand, while a push system initiates production in anticipation of future demand [12];
2. A pull system is one that explicitly limits the amount of WIP that can be in the system, while a push system has no explicit limit on the amount of WIP that can be in the system with the only limitation of the physical capacity [13];
3. In a pull system production is triggered by actual demands for finished products, while in a push system production is initiated independently of demands and usually based on forecasting [5].

Following these definitions, it is possible to model any (discrete) production flow as a composition of different entities (machines, buffer, etc.) that can be described with Push/Pull logic. In this context, different workstations, denoted by WSi,i=1…,n separated by buffers of finite capacity, denoted by Bi, i=1,.., n. can represent a basic production system.
In the picture below, an example of a process is shown. Here, 2 preparation lines join in an assembly station and then get the finished parts be stored in a warehouse. Intermediate buffers decouple activity in the upstream workstations from the downstream final assembly station.

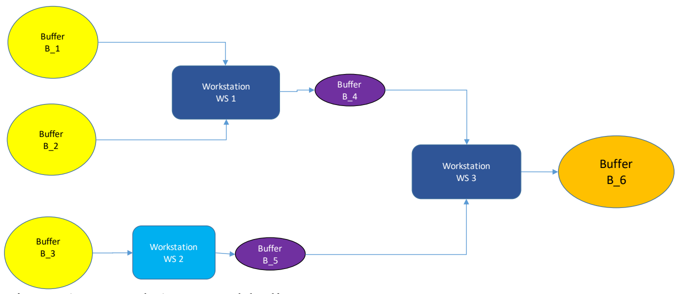


Figure 1 Example1: Assembly line

We can so state that a generic workstation consists of a machine, that can be classified (see Figure 2 – on the left) as:

1. Push-Push Machine
2. Pull-Pull Machine
3. Push-Pull Machine
4. Pull-Push Machine

When the Push-Push Machine finishes the part, it pushes it to the downstream station or buffer and starts working on a new part only when it is pushed from its upstream machine $WS_{i-1}$. If no part is available there, the machine is idle and the production is on hold.

Instead, when Pull-Pull receives a request from downstream workstation $WS_{i+1}$, it pulls the request to upstream workstation $WS_{i-1}$ for providing a component. Once it finishes the product it releases it to the downstream requestor and waits for another request.

The third machine receives and works a part pushed by the upstream workstation $WS_{i-1}$, but it releases that part only when it receives the request from downstream workstation $WS_{i+1}$.
Instead, the forth has the opposite behaviour: it requires a part as soon as it is available and it pushes the part as soon as that part is finished to the downstream entities.

In Figure 2 arrows specify direction of the operation trigger.

All four machines types have the maximum capacity of 1 product, it means that, before starting work a new product, the previous one needs to be released to the next stage.
A fifth class of machine behaves as a Pull-Push, but instead of requesting a product to a single machine, it requests products to two entities. Such machine implements the joining of two components in a new product, allowing the assembly of whatever complex products. This machine takes in 2 components that are "utilized" at the end of the machine cycle and "creates" the new component.
All 5 machines types operate with the following parameters:

- Operational time: the working time to transform/assemble products;
- Setup time: based on the product type, a setup is executed every time the production lot changes;
- Worked products: each machine can be specialized to work only a limited set of products;
- Optionally can be specified a failure rate to implement unavailability due to fails and associated repairing cycle.

All above entities implement a production phase characterized by an optional random variability of cycle duration.
In the same way, it is possible to represent 4 buffer typologies (see Figure 2 – on the right), as:

1. Pull Buffer
2. Push Buffer
3. Pull-Push Buffer
4. Kanban Buffer

The first Buffer describes a raw parts buffer that provide material when it receives a request and for this reason is called "Pull", it is the first entity of a production process. Vice versa, the Push Buffer implements a finished goods warehouse or a buffer with extracted items, it collects parts each time a Push-x workstation terminates production.
The last two buffers can be used to model the Interoperational buffers. The difference is that Pull-Push Buffer is a passive intermediate entity receiving components and providing parts while Kanban buffer is an active instance able to autonomously pull items from upstream machine, in order to keep a specific inventory level. This component is call Kanban buffer to underline its role in a pull system based on upwards propagation of the demand.
All four buffers are characterized by a given maximum capacity.

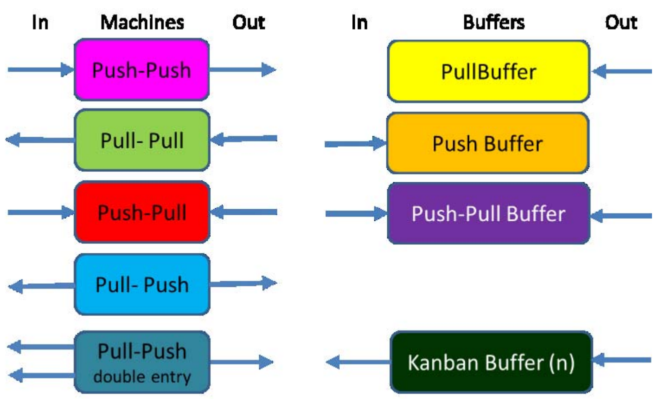


Figure 2 Production standard entities

## III. System Components Aggregation

A generic discrete production system can be represented aggregating and interconnecting a different set of the entities described in the previous chapter, specifying for each one the up-stream station (s) or the down-stream(s) entities. In doing that, it is necessary to respect compatibility rules that take into account how each entity works (i.e. input and output). Table 1 describes the possible coupling of the different production machines / buffers described so far.

Table 1 Aggregation Table

| | Downstream --> | Pull - Pull | Push-Pull | Pull-Push | Push-Push | Pull Buffer | PushPull Buffer | Buffer Push | Kanban Buffer |
|---|---|---|---|---|---|---|---|---|---|
| Upstream | Pull-Pull | Yes | No | Yes | No | NA | No | No | Yes |
| | Push-Pull | Yes | No | Yes | No | NA | No | No | Yes |
| | Pull-Push | No | Yes | No | Yes | NA | Yes | Yes | No |
| | Push-Push | No | Yes | No | Yes | NA | Yes | Yes | No |
| | Pull Buffer | Yes | No | Yes | No | NA | No | No | Yes |
| | Push-Pull Buffer | Yes | No | Yes | No | NA | No | No | Yes |
| | Push Buffer | NA | NA | NA | NA | NA | NA | NA | NA |
| | Kanban Buffer | Yes | No | Yes | No | NA | No | No | Yes |

It is also important to consider the possibility to have different machines working in parallel (e.g. to increase throughput of an operation) or to have multiple input or output. To this aim, it is important to consider that "*-Push" entities admit only one defined output, so "Push-*" machines cannot work in parallel. For the same reason, "*-Pull" entities cannot work in parallel, because downstream entities ("Pull-*" entities) need to specify a single source. In the following Table 2, the different options for multiple sources, drain or parallelism of a machine/buffer are represented.

Table 2 Parallel operations table

| | N-source | N-Drain | N-Istances |
|---|---|---|---|
| Pull-Pull | N | Y | N |
| Push-Pull | Y | Y | N |
| Pull-Push | N | N | Y |
| Push-Push | Y | N | N |

Based on the rules stated above, it is possible to simulate the behaviour of different discrete production processes, including linear, synthetic and analytic processes (like in Figure 1). In implementing a process, it is possible also to insert a retrofit loop to represent reworking. In the same way, it is possible to insert intermediate ”Push” buffers to store, for example, scrapped material.

## IV. THE COMPONENTS IMPLEMENTATION

In the implementation of the above referenced nine functional entities, we described their behavior with the Finite States Automata approach. In fact, as described in [14], a generalized sequential logic system that can be described by a number of output (n,o) which depends on the present and the past values of the input (n,i) can be formalized as a Finite State Machine (FSM).
Therefore, each production standard entities, introduced above, can be considered as a mathematical abstraction that describes all the states representing each possible situation in which these entities may ever be and all inputs and outputs with defined events. Events are also utilized to synchronized internal operations like end working or fault/repair.
To this aim, both standards values for Status and Events has been identified. They can trigger any state transitions, and the different behaviors each entity can have for any state transition. In this model based on Moore Machine implementation [15], the system behaviour is as a sequence of transitions that move the system through its various states [8]. From this, it is needed to identify several key characteristics of the system that have to be modeled with a finite state machine. First of all, the system has a particular initial state, it must be describable by a finite set of states and it must have a finite set of inputs and/or events that can trigger transitions between states [14]. For each machine a set of possible states and input/output/internal events have been identified:

TABLE 3 STATUS LIST

| Status Meaning | |
|---|---|
| Sx0 | Idle |
| Sx1 | Wait for product |
| Sx2 | Working |
| Sx3 | Unloading Product |
| Sx4 | Setup |
| Sx6 | Failure and repair |

TABLE 4 EVENTS LIST

| Events Meaning | | |
|---|---|---|
| E0 | Start/Stop | IN |
| E1 | Push product downstream | IN/OUT |
| E2 | OK – Acknowledge | IN/OUT |
| E3 | KO - Reject | IN/OUT |
| E4 | Ask product upstream | IN/OUT |
| E5 | End production cycle | INTERN |
| E6 | Failure | INTERN |

The following TABLE 5 represents what are the In/Out Events each entity can generate and manage.

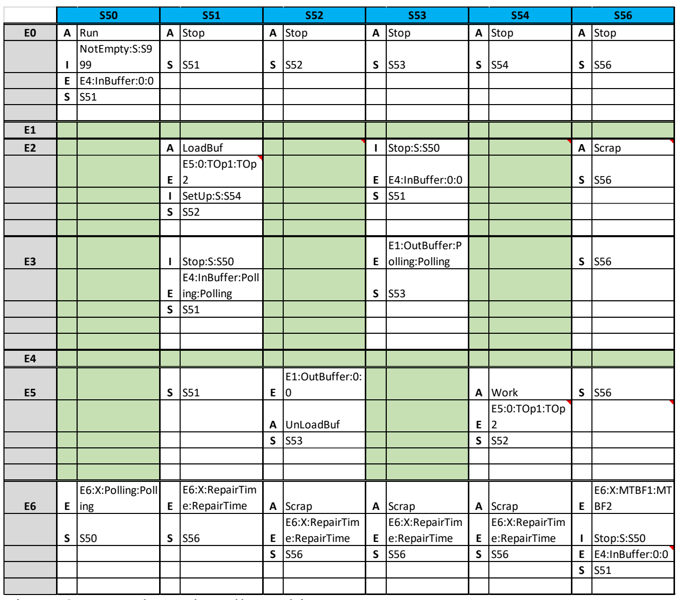
TABLE 5 EVENTS MANAGEMENT

| | Upstream | | | | Downstream | | | |
|---|---|---|---|---|---|---|---|---|
| **Events** | **E1** | **E2** | **E3** | **E4** | **E1** | **E2** | **E3** | **E4** |
| Pull-Pull | **--** | **In** | **In** | **Out** | **--** | **Out** | **Out** | **In** |
| Push-Pull | **In** | **Out** | **Out** | **--** | **--** | **Out** | **Out** | **In** |
| Pull-Push | **--** | **In** | **In** | **Out** | **Out** | **In** | **In** | **--** |
| Push-Push | **In** | **Out** | **Out** | **--** | **Out** | **In** | **In** | **--** |
| Pull Buffer | **--** | **--** | **--** | **--** | **--** | **Out** | **Out** | **In** |
| PushPull Buffer | **In** | **Out** | **Out** | **--** | **--** | **Out** | **Out** | **In** |
| Buffer Push | **In** | **Out** | **Out** | **--** | **--** | **--** | **--** | **--** |
| Kanban Buffer | **--** | **In** | **In** | **Out** | **--** | **Out** | **Out** | **In** |

According TABLE 5 and according the Table 1, it is now possible to specify for each entity (machine or buffer) what is the upstream and the downstream entity(-ies) and the respective inter-relationships.
The description of how each functional component works is carried out, as we said at the beginning of this chapter, via a Finite State Automata representation. To implement a model the user is not due to know or modify the way each automata works, once the operational and interconnection parameters are properly set. For this reason, it is important to remark that user is not expected to know/edit automata coding.
Nevertheless, each automata can be implemented via a reduced instruction set composed by 4 instructions:

| | |
|---|---|
| S | Next Status (define the status transition) |
| A | Action execution (Actions are code fragments) |
| E | Event (internal or external) generation |
| I | Conditional status transition or event generation |

In Figure 3, the implementation of a standard module (Pull-Push Machine) following the FSA formalism is represented as a sample.
Please note that empty cells in light green in Figure 3 represent that a certain event is not admissible in a specific status and for such reason no need to specify actions is required.
Consistently with the FSA approach, it is possible to recognize a table with a 5-tuple (Σ, Q, q0, F, δ), typically used to describe each FSA [14].
The set of symbols representing input (Σ) to SA are collected in the leftmost column where the list of possible events (internal or external events) is shown. In the top row, the list of the current status Q= {S1, S2, Sn} and of the final status $F \subseteq Q$ representing the set of states of SA, is depicted. $q0 \in Q$ is the initial state (state at time 0 of Automata) and it is identified in correspondence of E0 event.



| | S50 | | S51 | | S52 | | S53 | | S54 | | S56 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E0 | A | Run | A | Stop | A | Stop | A | Stop | A | Stop | A | Stop |
| | I | NotEmpty:S:S999 | S | S51 | S | S52 | S | S53 | S | S54 | S | S56 |
| | E | E4:InBuffer:0:0 | | | | | | | | | | |
| | S | S51 | | | | | | | | | | |
| | | | | | | | | | | | | |
| E1 | | | | | | | | | | | | |
| E2 | | | A | LoadBuf | | | I | Stop:S:S50 | | | A | Scrap |
| | | | E | E5:0:TOp1:TOp2 | | | E | E4:InBuffer:0:0 | | | S | S56 |
| | | | I | SetUp:S:S54 | | | S | S51 | | | | |
| | | | S | S52 | | | | | | | | |
| | | | | | | | | | | | | |
| E3 | | | I | Stop:S:S50 | | | E | E1:OutBuffer:Polling:Polling | | | S | S56 |
| | | | E | E4:InBuffer:Polling:Polling | | | S | S53 | | | | |
| | | | S | S51 | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| E4 | | | | | | | | | | | | |
| E5 | | | S | S51 | E | E1:OutBuffer:0:0 | | | A | Work | S | S56 |
| | | | | | A | UnLoadBuf | | | E | E5:0:TOp1:TOp2 | | |
| | | | | | S | S53 | | | S | S52 | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| E6 | E | E6:X:Polling:Polling | E | E6:X:RepairTime:RepairTime | A | Scrap | A | Scrap | A | Scrap | E | E6:X:MTBF1:MTBF2 |
| | S | S50 | S | S56 | E | E6:X:RepairTime:RepairTime | E | E6:X:RepairTime:RepairTime | E | E6:X:RepairTime:RepairTime | I | Stop:S:S50 |
| | | | | | S | S56 | S | S56 | S | S56 | E | E4:InBuffer:0:0 |
| | | | | | | | | | | | S | S51 |
| | | | | | | | | | | | | |

Figure 3 Example Push-Pull Machine Automata

Finally, the transition function (δ) is represented within the row and column intersection with three symbols:

A –Action Execution (a routine written in the platform language)

E – An internal or external (to another automata) event is generated

I – in the case of trigger event leads to a hypothetical condition; the two potential conditions are the eventual change of status or the generation of another event

The final implementation of a complex production system it so represented by a number of automata, each one representing a standard component, acting as autonomous agents and interacting each other. In the two platform prototypes implemented was also possible for automata to interact with physical world via interfacing libraries. [17]

## V. An Implementation Example

The industrial real case has been implemented within the EU-project PERFoRM ([16]) and it concerns to refrigerators and freezers production.

The real production process is composed of a production line made by different machines and buffers and it is represented in Figure 4. Following the production flow, the process starts with raw part Buffer (Buffer-Pull); it continues with Welding and the Degreasing workstations (Pull-Pull Machines) and with two parallel Push-Pull Machines (Primer workstation) and it finishes with New furnace station, which feeds the Buffer 4.

This process is designed for satisfying the overall external demand that is equally distributed among 4 different product types. For this reason, the last part of the line has been modelled with 4 different Pull-Push Machines, which provide their products to 4 finished goods buffers (Push Buffer). These are the crucial parts of the line, as, aiming to follow the market demand, they establish the production pace and the overall production demand is pulled to upstream machines.

For this reason, the machines and the inter-operational buffers are implemented respectively with Pull-Pull Machines and Kanban Buffers to achieve a "pulled" process. We can observe that machines 4 and 5 working in parallel and the following Push Buffer 19 implement a working center with same characteristics.

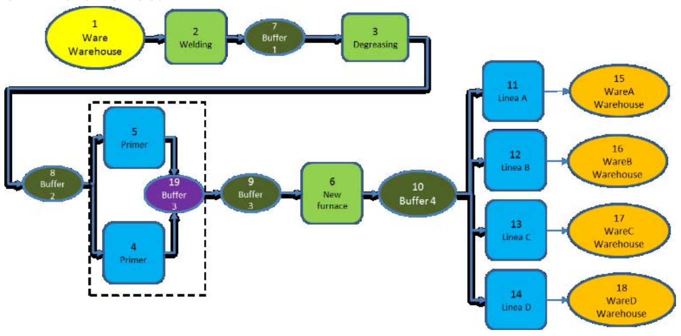


Figure 4 Use Case Industrial Plant

The interactions among the different entities, instantiated with the automata formalism, have been instantiated defining the initial state for each machine (this data characterizes the Automata Type among the 9 we presented), the startup events for each machine to trigger the operations, the identification of downstream and upstream entities for each module to establish connections and interactions. Furthermore, the production parameters explained in the chapter II have to be specified or as a characteristic for the machine or specific for the product type, while the max Capacity is the initial condition to be declared for buffer implementation.

This example was implemented in a MS-Excel/VBA based platform taking less than half an hour and launching it, we were able to get useful indications on buffer sizing and line balancing to dimension operational parameters for further analysis. If it is needed, a reconfiguration of the model for simulating different condition, it would take minutes.

## VI. Conclusions

This paper has proposed a specific approach aiming at providing a complete modelling and simulation process of a production system and it can be represented according to the schema depicted in Figure 5.

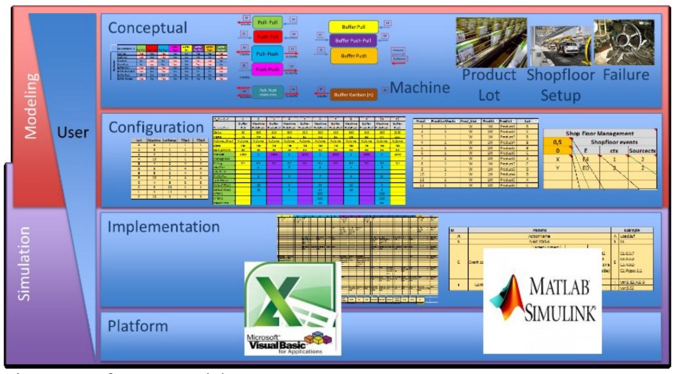


Figure 5 Reference model

In particular, from this figure it is possible to identify different steps.

At the top, the conceptual and configuration phases described in Chapters II and III are reported. In fact here, the specific production line needs to be modeled according to the aforementioned rules. To this aim, limited technical competences of the implementation platform, but a significant knowledge of the analyzed industrial context are required. The third level is described in Chapter IV and it refers to the implementation of components utilizing the Finite States Automata approach. It is not assumed to be modified by user even if modifications to the way a specific automata works are possible and can be carried out to implement specific extensions (e.g. human intervention, preventive maintenance, etc.). The lowest level deals with the actual implementation of the platform. Currently 2 prototypes are implemented: one on MS-Excel/VBA and the second on MATLAB-Simulink [17], showing its full adaptability to any production context.The platform is totally transparent from the functional standpoint to other levels.

## ACKNOWLEDGMENT

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 680435.

## VII. REFERENCES

[1] M. C. Fu and F. W. Glover, “Simulation optimization: a review, new developments, and applications,” in Winter Simulation Conference, 2005, no. 1, pp. 83–95.

[2] S. G. Platzman, L. K., “Simulating computer integrated manufacturing systems: how to model what traditional methods force you to ignore,” in IEEE International Conference on Systems, Man, and Cybernetics. IEEE, 1986.

[3] S. Ruiz-mier and J. Talavage, “Hybrid paradigm for modeling of complex systems,” Simulation Council, pp. 135–141, 1982.

[4] H. S. Ismail, V. S. Tey, L. Wang, and J. Poolton, “A UML Approach for the Design of Reconfigurable Manufacturing Simulation Models,” in IEEE IEEM, 2011, pp. 1690–1694.

[5] Liberopoulos, G. (2013, May). Production Release Control and the Push/Pull and Make-to-Order/Make-to-Stock Distinctions. In H. Tempelmeier, H. Kuhn and K. Furmans, eds., Proceedings of the 9 th conference on Stochastic Models of Manufacturing and Service Operations, Kloster-Seeon, Germany (pp. 113-120).

[6] B. Yu and J. Harding, “A reusable enterprise model,” Int. J. Oper. Prod. Manag., vol. 20, no. 1, pp. 55–69, 2010.

[7] Nance, R. E. (1996). A history of discrete event simulation programming languages (pp. 369-427). ACM.

[8] Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. The knowledge engineering review, 10(02), 115-152.

[9] B. Campana, G. Cimatti, “Measures and methods for a new taxonomy in manufacturing enterprises,” in Procedia CIRP, 2015, p. 287–292.

[10] Klaas, Thorsten. "Push-vs. Pull-Concepts in Logistics Chains." Proceedings of CEMS Academic Conference, Louvain-la-Neuve. 1998 pp. 1–14.

[11] R. S. Schonberger, Japanese Manufacturing Techniques: Nine hidden lessons in semplicity. Simon and Schuster, 1982.

[12] S. Nahmias, Production and Operations Analysis, 6th ed. Boston: McGraw-Hill, 2009.

[13] González-r, P. L., Framinan, J. M., & Pierreval, H. (2012). Token-based pull production control systems: an introductory overview. Journal of Intelligent Manufacturing, 23(1), 5.

[14] F. Boschi, G. Tavola, M. Taisch, “A description and analysis method for reconfigurable production systems based on Finite State Automaton environments,” in Service Orientation in Holonic and Multi-Agent Manufacturing, 2016, pp. 349–358.

[15] J. Hopcroft and J. Ullman, “Finite Automata,” in Introduction to Automata Theory Languages and Computation, S. Rivas, Ed. Pearson Education, 1979, pp. 37–81.

[16] PERFoRM – Production harmonizEd Reconfiguration of Flexible Robots and Machinery, “http://www.horizon2020-perform.eu,” 2016.

[17] Simulink, Matlab, and M. A. Natick. "The mathworks." (1993)” .