PETRI NET MODELS OF AGENT-BASED CONTROL OF A FMS WITH RANDOMLY GENERATED RECIPES

Adamo Castelnuovo
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio, 34/5- 20133 Milano (Italy)
castelnuovo@elet.polimi.it

Luca Ferrarini
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio, 34/5- 20133 Milano (Italy)
ferrarini@elet.polimi.it

Abstract

The paper addresses the problem of modelling a FMS with random recipes and controlling it with an agent-based architecture. One of the features of the system are the random recipes, i.e. recipes that are unknown a priori for the control system, which represent a high flexible-routing class of FMS. In particular, a PN model able to represent both the mechanism of generation of random recipes and then the execution of such randomly generated recipe is shown. The obtained formal model is then studied to develop suitable control policies, e.g. against deadlocks. Then, such a model, is implemented in one of the most popular agent platforms, JADE (Java Agent Development Framework).

1. Introduction

Flexible Manufacturing Systems are typically composed of machines, such as machining centres, of loading/unloading stations, assembly and quality control stations, interconnected by a transport system [8, 14]. During the last years, the growing complexity of these plants has increased the efforts in designing and implementing suitable coordination control functions. A significant step to achieve this goal has been made using agents.

Agents have been created as the natural extension of objects and different software platforms have been generated to support the set of their features: one of them is JADE [1], a Java Agent DEvelopment Framework, firstly conceived for telecommunication purposes.

Agents are software components characterized by autonomy, procreativity and sociality:

- autonomy: agents control their own action, they can even take decision about their behaviour;
- procreativity: agents can both react to external input and take initiative;
- sociality: they co-operate with other agents in order to pursue a common aim [7].

They have been applied to a wide variety of problems (data-mining, traffic control, distributed trade systems, economy, automotive, workflow optimization [4, 11]); their use in FMS automation is still an open research field, as for modelling, control and evaluation issues.

In particular, even if agents are nowadays present in many different industrial applications, systematic formal models describing their behaviour are not extensively described in the literature, mainly with respect to their flexible features. In this context worth mentioning is the work [5], where Petri nets model the structure for the agent's message passing function while the functional logic is described by a first-order logic involving each elementary task the agent can execute. Paper [11] summarises a set of Technology Research Initiatives started to support the development of Key Enabling Technologies, without facing formal models. Work [3] addresses ADACOR, a holonic control architecture using high-level Petri net models for each type of holon class representing the dynamic behaviours of the manufacturing components, extending the one defined in PROSA [13]. Worth mentioning is even the work [2], where the holonic characteristics of the IEC 61499 model for distributed intelligent control are exploited to describe a general approach for dynamic and intelligent reconfiguration. Last but not least paper [9] models the behaviours of the intrabay automation systems using distributed agent oriented Petri nets, an extension of Petri nets obtained adding agent-oriented modelling components to the standard modelling method.

The mentioned approaches have the common feature that the behaviour of an agent is either directly defined using a high-level programming language like Java, or the model of the agent and of the agent system is not formally described, or the formal model refers to a portion of the system under control. With reference to the manufacturing system automation, a common behaviour of a product agent is characterised by the

0-7803-9402-X/05/$20.00 © 2005 IEEE

flexibility to define the sequence of operations according to a few constraints and a goal function, namely it has the possibility to decide different routes or different machine orders to visit.

In the present paper, the considered recipes are *randomly generated*, i.e. the recipes on a raw product are characterised by a sequence of a finite number of operations, say **H** (unknown *a priori*) to be executed in a fixed order. The control system is not aware, ahead of time, of the recipes of products entering the work area. This case represents the most flexible situation for the production system, and thus one of the most difficult for the control, since it could lead to potential deadlocks in the system, without any possibility of analysis *a priori*.

The present paper proposes a Petri net model of the behaviour of an agent system with randomly generated recipes. The major problem is how to design a formal model describing the behaviour of a FMS when the working sequences themselves are not given *a priori*. A first way to face this problem is the one followed in [5], while a second way is to model with Petri nets both the random generation of a working sequence and, then, its execution: the latter is the approach here chosen. The paper thus provides a contribution to agent systems modelling, namely its major advantage is to have a formal model to represent the flexible behaviour of agents, as it is typical in FMS systems. The obtained model is then used to better understand the system evolution and to try different control laws in order to force a given closed-loop behaviour for the controlled system, organized in an agent fashion.

The present paper is organized as follows. The analyzed case study is described in Sect. 2. Sect. 3 illustrates the basic Petri net models for the agent and Sect. 4 explains how such modules have to be connected with each other to obtain the overall behaviour. Sect. 5 proposes some solutions to the risen deadlock problems and a notice about the simulation output obtained by means of JADE. Finally, in Sect. 6, the results are summarized.

2. A case study

In the present section, the system under study is described. This is a laboratory plant that has been built, with completely different purposes, for the European Project TORERO [15]. Its configuration is an extremely common for FMS with parallel machines. The system is constituted of the following components (see Fig. **1**):

- Load/unload station
- Part shuttle
- Six-place part storage (buffer)
- Work machines (1 to 4)

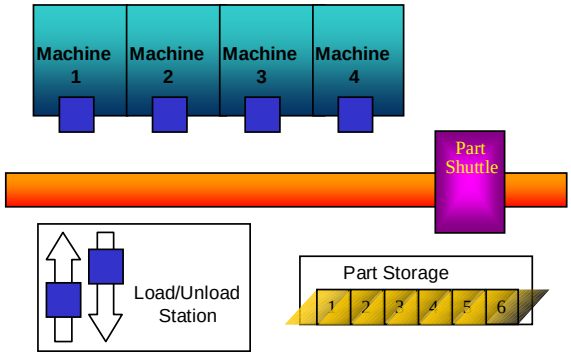


Figure 1. System layout

The behavior of the system operation can be summarized with the basic following rules:

- the part is constituted of a peg onto which a set of **n** rings will be placed (the placement is perfomed by the machines);
- $1 \leq \mathbf{n} \leq 4$, but the value of **n** is unkown *a priori*, for every part entering the system;
- in case a machine is busy, the next product which should use it is carried into and stored in the part storage;
- upon completion of its own recipe, the part is carried by the shuttle to the unload station.

As it can be seen, the peculiarity of this system is that any part entering it can be worked out with any possible sequence of operations (i.e. “visiting” any machine), from a minimum of one operation up to a maximum of four. In addition, the information on the recipe of a new product entering the system is not known *a priori*, by any “intelligent” component of the control system, i.e. there is no supervision at all. This follows the idea of agents, which encapsulate any behavioral rules, thus giving rise to a completely modular system (any agent can be removed if the corresponding physical system is removed or breaks down) but also to a completely dynamic system where even the sequence of operation is known, and no supervision is possible. As said in the introduction, such a dynamically reconfigurable system is quite new in FMS and definitely new in Petri net modelling, where product recipes are usually modelled before the system is controlled.

A description of the different components of the issued system follows.

2.1. The load/unload station

This station has two operating places, one for the load and one for the unload: in the load place a new part from outside arrives at a fixed frequency. Then, the shuttle, when ready, fetches it and bring it to a machine or to the buffer. When a part is finished, it is brought by the shuttle to the unload station, where it is automatically removed to the outside.

2.2. The shuttle

The shuttle can go back and forth along a unique direction. The admissible movements are:

- to the load station, if a new part is ready;
- to a machine, if this is available;
- to the buffer, if this is available and if the machine is not available to the unload station, when a part is finished.

2.3. The machines

Each machine task consists in placing a colored ring onto the peg and it is considered every machine has a sufficient number of rings to complete the working cycle. As the system uses just monochromatic rings every machine possesses only one color. Furthermore the issued system contemplates the chance to have more than one machine with the same color, without whatever connection with the physical layout. As no queue system is provided a machine can accept to work a part just if it is not working on another part.

2.4. The buffer

The buffer has just the purpose to keep a part during its wait for the requested machine (or better, a machine with the requested color) turn available. It has a six-place capacity.

2.5. The recipes

Recipes are dynamically created, as said, when a raw piece enters the system. Recipes are created as a set of operations to be done; in some cases, an operation can be performed also by more machines, thus giving rise to flexible routings inside the dynamically generated recipe. In addition, two main types of recipes specification are foreseen. The first one is called “fixed-order recipe”. In this case the recipe contains the exact sequential order that must be followed by the product. The second one is the so called “any-order recipe”, where the recipe specifies only a set (not a sequence) of operation, that can be executed in *any* order. Such types of recipes are schematically represented in Fig. 2, where the recipes are represented as the set of colours at the base of the peg (raw product).

The considered system works only on randomly generated fixed-order recipes and each operation can be performed by just one machine. Namely, at each step the next working operation to be executed is randomly chosen; then either the corresponding machine is available, or it is busy and the piece is led to the buffer until the machine turns available (availability of at least a free place in the buffer is checked in advance). So there is no chance to execute any operation on another machine if the requested machine is not free or it broke down.

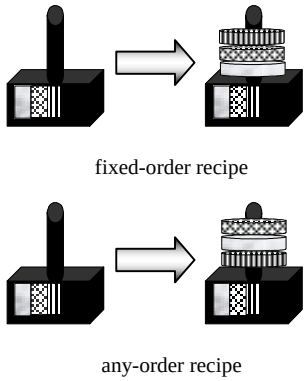


Figure 2. Fixed-order and any-order recipes

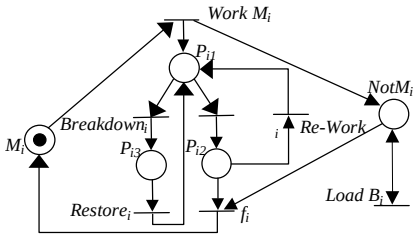
3. Basic Petri net modules of agents

Multi-Agents systems and agent technology can offer distributed intelligent control functions. This makes them a significative approch to develop and implement the components of intelligent manufacturing systems: each physical or abstract manufacturing resource is mapped into an agent [12]. Based on this approach, consider the Petri net modules representing the behavior of the previously described components of the system. For the sake of readability and because of the low contribute to the overall system the interaction with the shuttle has been omitted. The shuttle is actually an extremely important component, since it is shared among the different products and machines, and since its speed influences the overall system performance (as discussed in Sect. 5). However, the rule that will be forced is that each time the shuttle is used it must also be relased, i.e. before acquiring the shuttle, the conditions for its release must be unconditionally met. Consequently, form the logical point on view, for example for the allocation of resources or deadlocks problems, it has no influence at all on the overall system behaviour.

The models shown in the present section will be represented with labeled Petri nets. In particular, for any system components there will be a suitable module as well as for the random recipe, which finally will be composed (by both full synchronous composition (||) and interleave composition (|||) [6]) to obtain the overall system model.

Fig. 3 shows the module $\mathcal{M}_i$ for the machine. When a machine is available (one token in place M_i) whatever recipe requiring such a machine can start working on it (transition *Work* M_i). When work is over, transition *End* M_i fires and the workpiece stays into the machine until f_i (free machine M_i) fires in its turn and M_i becomes available again. Since an operation could fail during the work, the accident of machine breakdown has been considered: $Breakdown_i$ fires and the operation can be resumed just when the machine is restored (transition $Restore_i$).

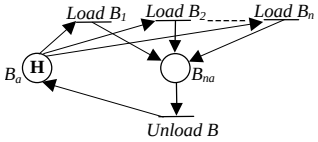
A workpiece requiring to be carried out by M_i can be brought to the buffer just if M_i is busy (place $NotM_i$ marked). The f_i transition is introduced with the purpose to release the machine M_i just when the workpiece actually leaves it. Transition *Re-Work* fires when a recipe has terminated an operation on M_i and requires to use it again.



Work M_i	Start working on M_i
End M_i	Stop working on M_i
f_i	M_i turns available
Re-Work	New operation on M_i without releasing it
Load B_i	Bring into the buffer since M_i is busy
$Breakdown_i$	M_i breaks down
$Restore_i$	M_i has been restored

Figure 3. Machine module $\mathcal{M}_i$

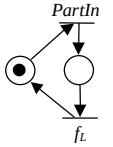
Fig. 4 points out the module $\mathcal{B}$ for a buffer with a **H** places capacity. Whenever a recipe requires a machine that is busy the workpiece is loaded into the buffer (transition *Load* B_i) so that the number of free places in it (place B_a) is decremented. When the requested machine turns available the workpiece is brought from to buffer to the machine and the number of free places in the buffer is incremented (transition *Unload B*).



Load B	Bring into the buffer
Unload B	Bring from the buffer to the required machine

Figure 4. Buffer module $\mathcal{B}$

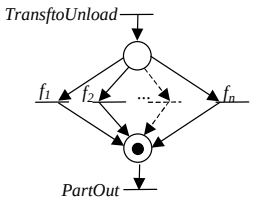
Fig. 5 shows the module $\mathcal{L}$ for the Load station. When a new workpiece requires to get into the system and be worked, transition *PartIn* fires. No other piece can be introduced until the Load station turns available by transition f_L firing.



PartIn	Bring part into the system
f_L	Free Load station

Figure 5. Load station module $\mathcal{L}$

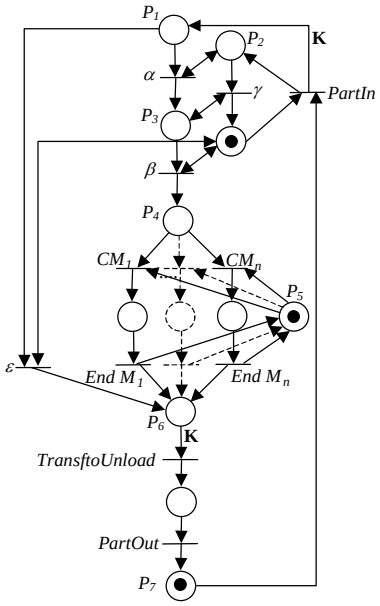
Fig. 6 points out that one for the Unload station. After the last working, a piece requires to be brought into the Unload station (transition *TransftoUnload*), the machine it occupies is released (transition f_i) and the workpiece is unloaded (*PartOut*).



TransftoUnload	Require to be brought out of the system
f_i	Machine M_i released
PartOut	Workpiece unloaded

Figure 6. Unload station module $\mathcal{U}$

Fig. 7 illustrates the module for the generation of the random working sequence of no more than **K** operations (with the same meaning as **n** in the laboratory plant case). It is constituted of three parts: in the first the piece is loaded into the system and the number of operations to be executed is set, in the second the necessary machines are randomly chosen, one at a time, and in the last transfert from the machining system to the Unload station is represented.



PartIn	Workpiece loaded, start random generation
α	Add operation
β	(Functional transition)
γ	Stop random generation
ε	(Functional transition)
CM_i	Choose machine M_i
End M_i	Stop working on M_i
TransftoUnload	Require to be brought out of the system
PartOut	Workpiece unloaded

Figure 7. Random recipe generation module $\mathcal{G}_j$

When transition *PartIn* fires the worpiece is loaded, **K** tokens are put into the place P_1 and one token is put

into P_2. Transition γ can not fire until at least one token is into P_3, namely the recipe contains at least one operation. Transition α can fire no more than **K** times. When γ fires P_3 holds the number of operations in the recipe. Before a piece is unloaded all tokens are moved from P_3 to P_4. Place P_5 guarantees that operations are executed one at a time. When all operations on a workpiece have been completed P_6 contains **K** tokens (otherwise ε fires independently) and the piece requires to be unloaded (transition *TransftoUnload*). When the last recipe occupied is released it is brought out of the system (*PartOut*), the initial state is fully recovered and if transition *PartIn* is enabled a new workpiece can be put into the system.

Each recipe execution model can be obtained as shown in Fig. 8 juxstaposing modules representing both the working on the piece and the transfert to the buffer (squared sub-net) and introducing additional Petri net's elements. When the machine M_i is chosen CM_i fires marking P_{Ai}. If M_i is available (place M_i in Fig. 3 marked) *Work* M_i fires and P_{Bi} is marked, otherwise, if there is at least one free place in the buffer (one token or more in place B_a of Fig. 4) *Load* B_i fires and P_{Ci} is marked. In both cases if the issued operation is the first (*P'* marked) f_L fires releasing the Load station; in the other cases f_i fires releasing the last machine the issued workpiece occupies. When the previous occupied resource has been released a token is put either in P_{Di} or in P_{Ei}, respectively.

If the workpiece has been brought into the buffer (P_{Ei} marked), it waits for M_i to turn available; *Work* M_i fires followed by *Unload B* and P_{Di} is marked.

The transition *Re-Work* offers the chance to repeat an operation onto the same machine without releasing it. The self-loop P_{Fi} *-Load B*, in every squared sub-net, guarantees that, whenever the next operation to be executed is on M_i again, the workpiece is not brought from M_i into the buffer. The arcs drawn from each place P_{Hi} to every transition f_i warrants the releasing of the machine the issued workpiece occupies and not a machine related to a different recipe. The arcs drawn from each transition f_i to every place P_{Fi} restore the initial marking, letting *End* M_i fire the next time an operation on M_i is over. When all operations to be executed are over one of the transitions *PartIn* can fire and a new workpiece can get into the system.

4. Composition rules of PN agents

Petri net is a DES modelling methodology describing events and conditions of the DES behaviors in terms of the casual relationships between events. Sometimes it is useful to compose models in order to obtain new models. The two fundamental operators literature given to compose (labeled) Petri nets are full synchronous composition (||) and interleave composition (|||). In the former common events between two processes can occur just together; if a common event can occur in one of the processes it has to wait until it can occur even in the other one. In the latter the previous constraint is no more present so that no events are synchronized between the two processes [6].

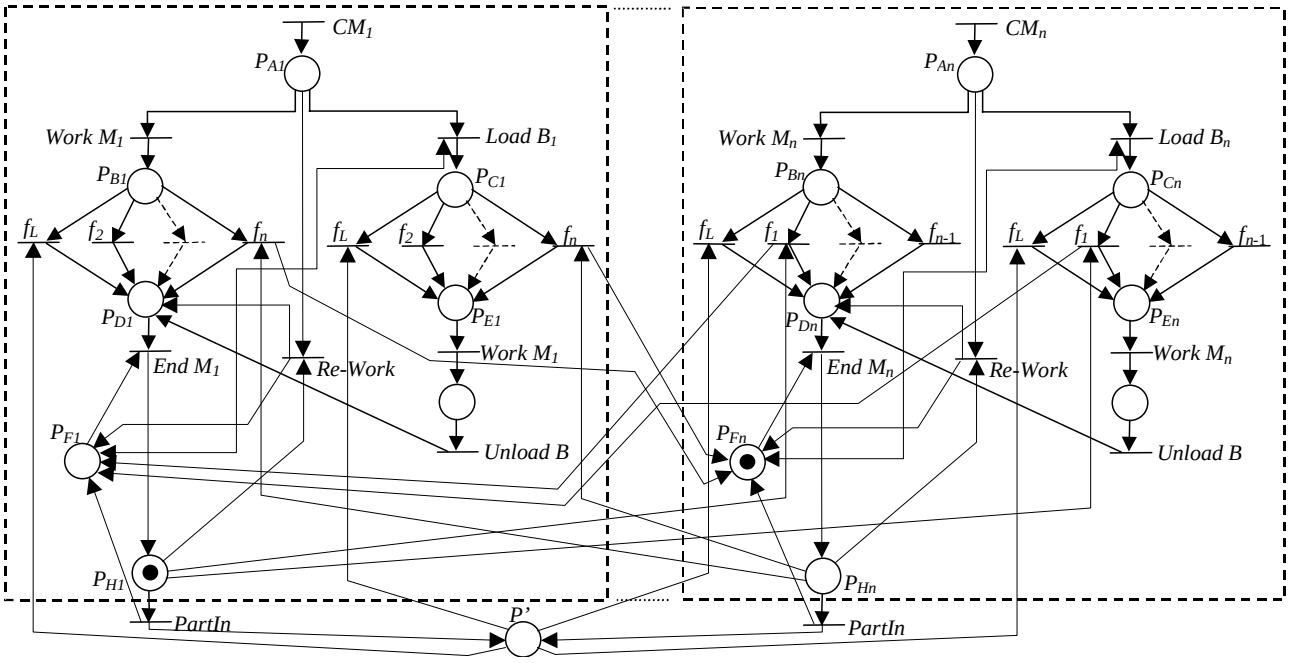


Figure 8. Recipe execution model $\mathcal{R}_i$

With respect to the case study issued in Sect. 2, the overall model $\mathcal{P}$ for a system endowed with **Z** recipes and **M** machines can be obtained by the following expression:

$$\mathcal{P} = (\;|||_{j=1}^{Z}\; ((\mathcal{R}_j ||| \mathcal{U}) || \mathcal{G}_j)) ||| (\;|||_{i=1}^{M}\; \mathcal{M}_i) || \mathcal{B} || \mathcal{L}. \qquad (1)$$

In (1) the interleaving operator composes every recipe model $\mathcal{R}_j$ with $\mathcal{U}$. Then the resulting model is fully composed with $\mathcal{G}_j$. The use of just the full synchronous operator would have led into an error: non-required synchronization between releasing events in $\mathcal{R}_j$ and in $\mathcal{U}$ (transitions f_i) would have risen. Models thus obtained are then justaxposed with the interleaving operator to constitute the set of the maximum number of recipes simultaneously developing in the system. As the set of machines and the buffer and the Load station are shared resources, the issued model is fully synchronized with them.

5. Control system design

The aim of the proposed study has been mainly the modelling of plant components by Petri nets which could easily represent the corresponding agent behaviours, the implementation of the agents in JADE for simulation purposes following. As it could be easily foreseen, the simulation of the system (1) described in Sect. 2, exhibits some deadlock states that can be solved introducing a suitable controller. The present section addresses this problem.

Different anti-deadlock policies can be conceived to pursue this aim, both optimal and sub-optimal. The former ones are out of the scope of the present paper, but they are subject of future investigation. Among the latter, the following simple ones can be described.

A workpiece can get into the system only if:

(a) at least **M** places in the buffer are free;
(b) condition (a) is verified, or the buffer is full or it has just a free place but all workpieces are on their last workings;
(c) condition (a) is verified, or the buffer is full or it has just a free place but all workpieces require to be worked by free machines.

Obviously, other less conservative but more complex solutions could be checked onto the system to obtain deadlock prevention. The proposed sub-obtimal ones are characterized by simplicity, robustness and can be easly implemented; on the other side they have the drawback to prevent some acceptable states from being reached. Different control methodologies can be adopted to introduce the solutions and the resulting controller can either be implemented as an arbitrator agent or it can be distributed into the present agents. For instance the design method based on supervisory control [10] can be adopted to satisfy condition (a). In particular, the mixed constraint (2), i.e. a constraint on both places and transitions, can be formulated:

$$m_{B_{na}} + (\mathbf{H}\text{-}\mathbf{M}) s_{PartIn} \leq \mathbf{M}(\mathbf{H}\text{-}\mathbf{M}),\ \forall \mathcal{R}_j,\ j=1, ..., \mathbf{Z}. \qquad (2)$$

It guarantees that in whatever recipe $\mathcal{R}_j$, transition *PartIn* can fire only if there are no more than **H**-**M** token in B_{na} (it contains the number of free places in the buffer), namely no more than **H**-**M** occupied places in the buffer (at least **M** free places).

The corresponding control Petri net is shown in Fig. 9 (above). Such a net will be syncronously composed with $\mathcal{P}$ to obtain the controlled model. An intuitive solution for the same control policy is given in Fig. 9 (below), where the new buffer module $\mathcal{B}'$ comprises the full control policy therefore acting itself as a controller, in a distributed environment.

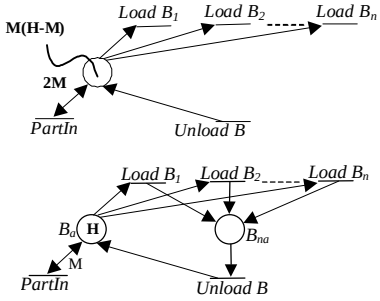


Figure 9. Control module obtained by supervisory control and intuitive control module $\mathcal{B}'$

The controlled system has been implemented in JADE and simulations have shown that no more deadlock states were still present. In order to verify deadlock avoidance, parameters have been suitably calibrated. JADE is not endowed with testing tools and cost functions, so the program has been created in order that data could be automatically saved. They are in such a way at anyone's disposal for analysis and performance investigation. In particular, some of the more interesting saved information are: working time related to each machine, number of occupied places in the buffer and time from loading to unloading of a workpiece. Due to space limitations, most evaluation results are omitted.

An example of the application of the previously mentioned sub-optimal solution (a) is given in Fig. 10. The simulation in JADE has been made with respect to the working of 150 workpieces, each one with a randomly generated recipe, and shows the average number of occupied places in the buffer versus the machine working time. This could be used as an index of the saturation degree of the system.

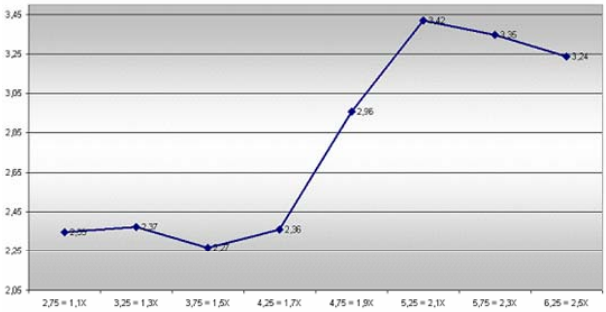


Figure 10. Average number of occupied places in the buffer vs. the machine working time

6. Concluding remarks

The paper has shown how to build formal models with Petri nets for agent-based flexible-behavior control systems. With the help of a case study, the functional characteristics of each component (machine, load/unload station, buffer and shuttle) have been addressed and the corresponding Petri net models have been provided, together with the module for the recipes' random generation and the recipes' execution. The overall model has been obtained by means of the full synchronous and interleave composition. Sub-optimal solutions to the problem of deadlocks has been mentioned and one of them has been implemented in the agent environment JADE for simulation purposes. Finally an example of simulation output data analysis has been given. Future directions include the introduction of fault tolerance by more machines performing the same operations, the management of any-order recipes, the full study of deadlock avoidance, the complexity analysis of the overall model with respect both to the number of machines and to the places in the buffer, the automatic translation of Petri net models in JAVA for JADE and the application to real-scale plants.

References

[1] F. Bellifemine, G. Caire, A. Poggi and G. Rimassa, "JADE White Paper", web site: http://exp.telecomitalialab.com, Vol. 3, no. 3, 2003.

[2] R. W. Brennan, M. Fletcher and D. H. Norrie, "An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems", *IEEE Transactions On Robotics And Automation*, Vol. 18, no. 4, pp. 444-451, 2002.

[3] A. W. Colombo, P. Leitão and F. J. Restivo, "ADACOR: A Collaborative Production Automation and Control Architecture", *IEEE Intelligent Systems*, Vol. 20, issue 1, pp. 58-66, 2005.

[4] S. M. Deen, *Agent-Based Manufacturing. Advances in the Holonic Approach*. Springer, 2003.

[5] W. Han and M. A. Jafari, "Component and Agent-Based FMS Modelling and Controller Synthesis", *IEEE Trans. on Systems, Man, and Cybernetics*, part c, vol. 33, no. 2, pp. 193-206, 2003.

[6] C. A. R. Hoare, *Communicating Sequential Processes*, *International Series in Computer Science*, Prentice-Hall Int., 1985.

[7] N. R. Jennings and M. J. Wooldridge, *Applications of Intelligent Agents. Agent Technology: Foundations, Applications and Markets*, Springer, pp. 3-28, 1998.

[8] S. B. Joshi and J.S. Smith, *Computer Control of Flexible Manufacturing Systems - Research and Development*, Chapman&Hall, 1994.

[9] C. Kuo, C. Wang and K. Huang, "Behavior Modelling and Control of 300 mm Fab Intrabays Using Distributed Agent Oriented Petri Net", *IEEE Transactions on Systems, Man, And Cybernetics*, part a, vol. 33, issue 5, pp. 641-648, 2003.

[10] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*, Kluwer Academic Publishers, 1998.

[11] R. Schoop, A. W. Colombo, B. Suessmann and R. Neubert, Industrial Experiences, "Trends and Future Requirements on Agent-based Intelligent Automation", *IECON '02*, vol. 4, pp. 2978-2983, 2002.

[12] B. Suessmann, A. W. Colombo and R. Neubert, "An Agent-based Approach Towards the Design of Industrial Holonic Control Systems", *Proc. On the 5th IEEE- IFIP Int. Conf. On Information Technology for BALANCED AUTOMATION SYSTEMS in Manufacturing and Services*, Cancùn, Mexico, 2002.

[13] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts and P. Peeters, "Reference Architecture for Holonic Manufacturing Systems: PROSA", *Computers In Industry*, vol. 37, pp. 255-274, 1998.

[14] M. C. Zhou and K. Venkatesh, *Modelling, Simulation, and Control of Flexible Manufacturing Systems*, World Scientific, 1999.

[15] Web site: http://www.uni-magdeburg.de/iaf/cvs/torero