Data processing from manufacturing systems to decision support systems: propositions of alternative design approaches

Mickaël Adam, Olivier Cardin, Pascal Berruet, Pierre Castagna

Data processing from manufacturing systems to decision support systems: propositions of alternative design approaches

M. Adam*, O. Cardin*, P. Berruet**
P. Castagna*

* LUNAM Université, IUT de Nantes – Université de Nantes, IRCCyN UMR CNRS 6597 (Institut de Recherche en Communications et Cybernétique de Nantes), 2 avenue du Pr Jean Rouxel – 44475 Carquefou
(e-mail: mickael.adam/olivier.cardin/pierre.castagna@univ-nantes.fr).
** Laboratoire en sciences et technologies de l'information, de la communication et de la connaissance, Lorient, France
(e-mail: pascal.berruet@univ-ubs.fr).

Abstract: With the increase of flexibility and production rates, the complexity of manufacturing systems reached a point where the operator in charge of the production activity control of the system is not able to forecast efficiently the impact of his decisions on the global performances. As a matter of fact, more and more Decision Support Systems (DSS) are developed, as much in literature or industrial applications. DSS have one common point: the initialization of their forecasting functionality is based on data coming from the manufacturing system. Furthermore, this feature is fundamental, as it has a direct impact on the accuracy of the forecasts. Considering the variety of input and output data, a data processing is necessary to adapt those coming from the manufacturing system. The aim of this paper is to present several design approaches enabling the integrator of a new manufacturing system to speed up the implementation, with the idea of automate and systematize the maximum design phases thanks the model driven engineering.

Keywords: Decision support systems, Data processing, Manufacturing systems, Model driven engineering, Model transformation, Observers.

1. INTRODUCTION

With the increase of flexibility and production rates, the complexity of manufacturing systems reached a point where the operator in charge of the production activity control of the system is not able to forecast efficiently the impact of his decisions on the global performances.

As a matter of fact, more and more Decision Support Systems (DSS) are developed, as much in literature or industrial applications. Each DSS has its own performance and hypotheses, but they have one common point: the initialization of their forecasting functionality is based on data coming from the manufacturing system. Furthermore, this feature is fundamental, as it has a direct impact on the accuracy of the forecasts.

As the data available on the system are generally not directly usable in the initialization of the DSS, a data processing is necessary to adapt the data (Fig.1). The major difficulty is that the entrance data are each time different, when those needed by the DSS are also each time different.

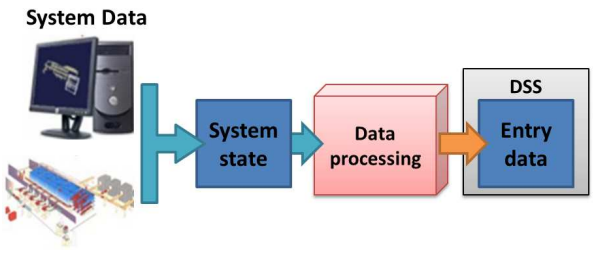

Fig. 1. Initialization of a Decision Support System

At last, the problem of response time is very important when dealing with short-term DSS. Indeed, as the forecasts are made on short horizons, the data processing has to be shortened so that the operator can make his decision as early as possible.

The aim of this paper is to present several design approaches enabling the integrator of a new manufacturing system to speed up the implementation. The idea is to automate and systematize the maximum design phases so that the variety of data at the entrance and at the exit of the data processing can be reduced. These approaches are mainly base on Model Driven Engineering (MDE).

First section of this paper shows several examples of DSS found in literature, in order to show the diversity of applications and data involved. Second section formalizes the problems of data processing. Third section suggests alternative architectures enabling an efficient data processing, according to the application aimed. Finally, last section exposes the design approaches suggested to help the implementation of the most complicated (but also the most efficient) solutions presented in third section.

2. MANUFACTURING SYSTEMS AND DSS

When a disruption occurs on a manufacturing system, it is generally necessary to take decisions. Many Decision Support Systems can be found in literature. This section introduces some of the most used and detail the processing of data needed to feed them.

Some kinds of DSS use online simulation. The performance of online simulation, base on short term decisions calculated

by discrete-event simulators, is known for several decades. It has been recently experimented on large scale systems via observers (Cardin *et al.*, 2009). On specific problems, (Mahdavi et al. 2010a), (Mahdavi et al. 2010b) use for initialization a list of past events and a forecasted list of probable future events. In (Chong *et al.*, 2003), when a disruption occurs, several new schedules are calculated. A discrete-event simulator evaluates them before choosing the most adequate for the decision.

Another kind of DSS refers to the field of artificial intelligence via fuzzy logic (Mok, 2009) (Garavalli, 1999) or artificial neural networks. The applications are wide, from the definition of a sensors fault tolerant control (Magdy, 2009) to the prediction of drill wear from thrust force and cutting torque signals (Yang et al., 2009). These systems need a learning phase, generally obtained with an observation of the states of the system along the production (Pierreval, 1992).

This short study intends to show that the different DSS encountered in literature need various data coming from the manufacturing system. Furthermore, the set of data is very variable, can be huge, and the precision required is various (mean load of a buffer / identification of the products present in the buffer, for example).

3. THE PROBLEM OF AVAILABILITY OF DATA FOR DSS INITIALIZATION.

This section delineates the problematic. To illustrate this, a simple case study is presented, based on a conveying system with a workstation in derivation. This example will be later used to illustrate the concepts which will be developed in section 4.

3.1 Introduction

As shown on figure 3, the physical structure of the workstation is made of 2 conveyors, 3 sensors and 1 stopper. When a part reach sensor 1, and if workstation 1 is available, then stopper 1 let the part go and the divergence makes the part enter the station.

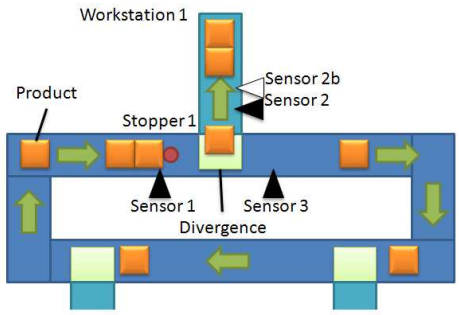

Fig. 2. The example of a workstation

3.2 Data in a manufacturing system

A manufacturing system is constituted of many data, of variable nature and localization. Among the most classical data, data coming from the sensors, coming from the actuators or data coming from the production activity control, such as the state of execution of the manufacturing orders, could be cited. It is theoretically necessary to retrieve the values of all these variables to define, at time t, the set $S_{system}(t)$, corresponding to the state of the system.

Inside this set, the subset of observable states $S_{obs}(t)$ contains all the values of the variables that can be retrieved at time t. These data can be located in the control devices (and thus generally easy to retrieve via protocols such as OPC, MODBUS, etc…) or in the MES database for example.

Non-observable states subset $S_{unobs}(t)$ contains all the variables which value cannot be retrieved at time t. When some variables are inside this subset at any time, some can be part of one set or another, depending on the considered date. The most classical example is the location of a part on a conveyor with a minimalist control. Its exact position is known when it is in front of a sensor; no data directly give the exact location of the part between two sensors.

The state of the system can thus be defined as the union of both the subsets previously defined:

$$S_{system}(t) = S_{obs}(t) \cup S_{unobs}(t) \quad (1)$$

With:

$$S_{obs}(t) \cap S_{unobs}(t) = \varnothing \quad (2)$$

3.3 Concurrent data representation in the control

Among all the data available on the control system, let us focus on the implementation of the resource sharing of workstation 1. Different classical implementations could indeed be considered.

Fig. 3 shows three different ways to code the authorization for a part to enter the station. These three examples are all written using the same IEC 61131-3 SFC language to facilitate the reading, but the reader should keep in mind in the following that the language could furthermore be different.

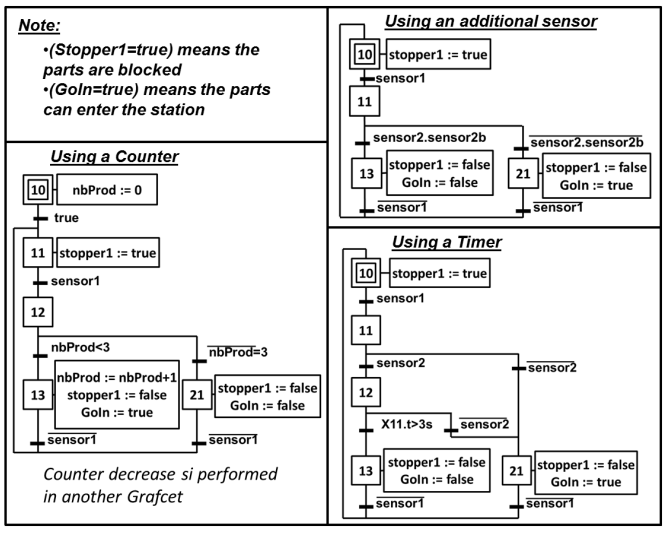

Fig. 3. Three ways of coding a workstation entrance

First way is to evaluate the activation delay of the entrance sensor. When a critical threshold is reached, the station is

considered as full. Second way, more classical, uses a counter, representing the number of part present in the station. The increment of the counter is represented; the decrement is located in the exit_station's program. Third way uses a second sensor (2b), located just after sensor (2). When both sensors are active at the same time, then the station is considered as full.

$S_{obs}(t)$ differs between the 3 examples, alternatively containing variables such as "sensor1", "sensor2", "sensor2b", "stopper1" or "nbProd".

If the DSS requires data about the state of the station's buffer ("full"/"not full"/"empty"), one variable ("nbProd"), two variables ("sensor2" and "sensor2b") or the activation time of a SFC step alternatively are to be retrieved. However, these data needs to be adapted to fit the need of the DSS.

So although these three ways of coding have the same behavior, they cannot enable the implementation of any DSS directly connected to the control.

3.4 Requirements of DSS and the need for data adaptation

This following example is based on a DSS, which is used when a part has to enter the station and the buffer is full. This DSS provides data to decide whether it is better to wait for the station to get available – thus blocking the other parts on the conveyors – or to stay on the conveyor and make one more lap before attempting again to enter the station – with a high level of risk for another part to have entered the station before. This kind of decision is relatively tricky, as it involves a lot of parameters.

To be able to provide the most accurate data, the DSS needs a set of data coming from the system for its initialization: this set is denoted $S_{needed}(t)$. This set depends on several parameters, such as the nature of the DSS, the time and the objective of the question asked to the DSS. Obviously, it does not correspond to the whole set of data composing the state of the whole system:

$$S_{system}(t) = S_{needed}(t) \cup S_{not\_needed}(t) \quad (3)$$

Considering the DSS of the example, it is obvious that the decision must be short-term. Thus, an important characteristic of the data adaptation between $S_{system}(t)$ and $S_{needed}(t)$ is the response time. Indeed, a long response time provides a bias in the prevision, which has to be reduced at a minimum value. As a matter of fact, if $S_{needed}(t) \cap S_{unobs}(t) \neq \emptyset$, then a reconstruction, potentially time consuming is needed.

3.5 Problematic

The last section clearly stands the three major issues that have to be faced when implementing a DSS on an existing control. Firstly, the list and the nature of the variables necessary for the initialization of the DSS are generally completely different for one DSS to another. Secondly, the correspondence with the data available in the system's state is difficult to establish. Finally, the response time for initializing the DSS is an important feature to be taken into account when implementing the DSS.

To face these issues, next section introduces and discusses several alternatives about data processing for DSS.

4. DATA PROCESSING TO INITIALIZE DSS

In this section, various architectures, and the associated data processing, are presented with the objective to initialize a DSS coupled to a manufacturing system. Indeed, according to the content of $S_{obs}(t)$ and the requirements of $S_{needed}(t)$, the architecture can be quite different.

The illustrations are based on the example presented in the previous section.

4.1 First solution: direct state transfer

The easiest solution (Fig.4) to implement is to initialize the DSS directly with the state of the variables contained in the control of the system $S_{obs}(t)$. The difficulty is to have a control and a DSS which deal with the same variables.

Indeed, the DSS cannot access variables value from $S_{unobs}(t)$, and cannot either reconstruct them, having no data about the evolution of the variables values before t. The previsions of the DSS are thus less precise, unless the control provides a lot of data, which is generally expensive to implement.

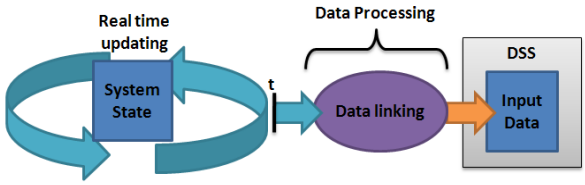

Fig. 4. Direct data transfer

As shown on Fig 4, a linking between data is necessary. This linking is made once, and enables relationships between the variables of the system and the variables of the DSS. It answers the concerns explained in section 3.3.

4.2 Second solution: Using the history of events

To get some data included in $S_{unobs}(t)$, a solution is to work with the history of the data of the system. This history is stored along the production, and processed when the DSS is initialized (Fig. 5).

The example of the parts on a conveyor is revealing. When the DSS is initialized, the last occurrences of the entrance and exit sensors are processed. Associated with the speed of the conveyor, the position of the items can be reconstructed with a simple regression. The content of the buffer in an accumulation conveyor can also be retrieved with the same data.

Obviously, the data linking is still necessary when parameterizing the connection between the system and the DSS. However, the data processing is also composed of the

state reconstruction, which can be long to process. As a matter of fact, this solution is not always compatible with applications for a very short-term DSS. Another problem remains in the amount of data to be stored during the life of the system.

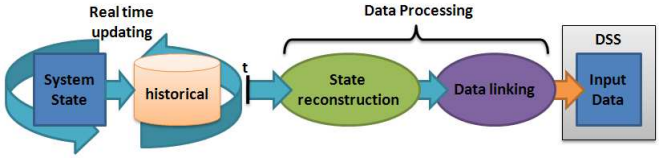

Fig. 5. Using a history of states.

4.3 Third solution: Using a generic state reconstructor

The third solution presented relies on the same idea, but tries to discard the problem of time consumption and data storage.

This solution (Cardin, 2007) enables to reconstruct a $S_{system}(t)$ as complete as possible of the system thanks to an observer, also called state reconstructor. The state of the observer can therefore be considered, at any time t, as the closest image of $S_{system}(t)$ possible.

The initialization of the DSS (Fig. 6) is thus similar to the first solution, based on a direct data transfer between the state of the observer and the DSS, with a very short data processing.

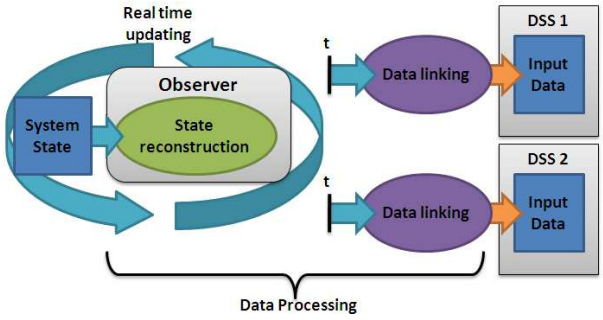

Fig. 6. Using a generic observer

This solution is very efficient, but two main issues remain:

1. The design of the observer is a difficult task, as the mechanisms of synchronization between the observer and the control are hard to establish;
2. The data linking is still present, as it is necessary to establish the relationships between variables each time a new DSS has to be implemented.

4.4 Fourth solution: DSS-oriented observer

In order to overcome this last issue, the idea was to integrate the linking of the data directly in the design phase of the observer. The objective is to go from the state of the observer considered, at any time t, as the closest image of $S_{system}(t)$ possible as it was the case before, to the state of the observer considered, at any time t, as the closest image of $S_{needed}(t)$ possible. Furthermore, this should decrease the complexity of design of the observer, as the number of variables should be decreased.

The method consists thus in integrating the model of the considered DSS data structure as the data structure of the observer (Fig.7).

The main issues of such a solution are:

1. A supplementary observer has to be designed each time a new DSS needs to be implemented on the system, as the observer are dedicated to specific DSS;
2. The design of the observer, although less time consuming because shorter, is even more complex as the data structure of the DSS has to be integrated.

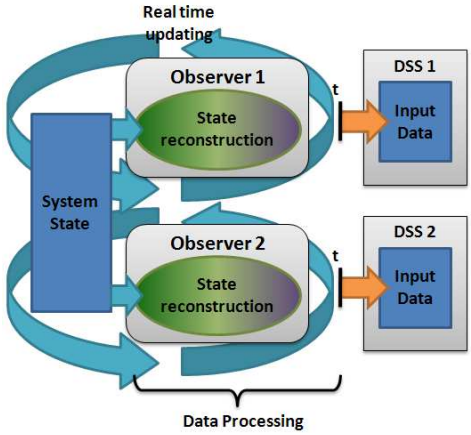

Fig. 7. DSS-oriented observer

4.5 Conclusion about these solutions of data processing from a manufacturing system to the corresponding DSS

As mentioned above, several ways are possible to deal with the identified problem, from the most simple to the most efficient. The main difference stands in the position of both the state reconstruction and the data linking. To evaluate the relevancy of implementation of one solution on a real case, four factors can be compared:

- The need of the chosen DSS for variables belonging to $S_{unobs}(t)$, and thus the need for state reconstruction;
- The compliance with the real-time requirements of short term DSS;
- The genericity of the approach;
- The complexity of design of the solution.

Table 1 classifies, for each of the suggested solutions, the level of these factors.

The levels of the factors are very dependant of the use that is meant for the DSS. However, it underlines that the solutions with the observers are interesting. Their major drawback remains the design complexity, which makes their implementation quite impossible at wide range. Next section presents design approaches proposed to facilitate design of both the generic and the DSS-oriented observers.

Table 1. Comparison of the solutions

Solution :	State reconstruction :	Real-Time compliant :	Genericity of the approach	Design complexity
Direct transfer	No	Yes	No	Very low
Direct transfer with historical reconstruction	Yes	No	No	Low
Generic Observer	Yes	Yes	No	High
DSS-oriented Observer	Yes	Yes	Yes	Very high

5. DESIGN APPROACHES

This section intends to present a generic design approach, declined in two versions depending whether it deals with a generic observer or a DSS-oriented observer.

These approaches are based on Model-Driven Engineering (MDE) and have the originality to systematize a global design, including control and state reconstructor. To ease the designer work, as many steps as possible are automated.

5.1. Designing a generic observer

This first approach was presented in (Adam *et al.*, 2011). The aim of the authors was to create a design flow, presented in Fig. 8, automating the generation of the control code and the observer.

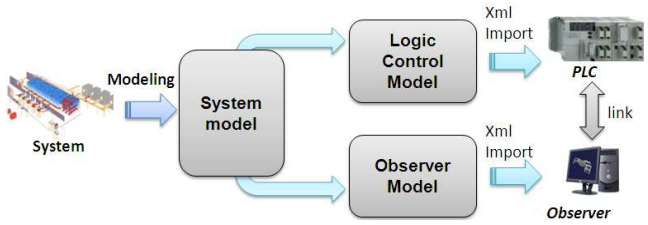

Fig. 8. Design flow of a generic observer

This flow enables the end user to generate a customizable observer, with all the links with the control already established, only based on the UML description of the conveying system, as described in (Lallican *et al.*, 2007).

Using the same standard as (Lallican *et al.*, 2007) is interesting, as it allows to reuse their results about control code generation. This parallelism of generation is the key point for an easy establishment of the links between the control and the observer.

Considering the example Fig 2, the designer first models the conveying system (Fig. 9). This model contains all the relationships between components, their position on the conveyor, the tasks they can perform and all the details on the implementation which can be useful (such as the PLC technology and the language that is wished for example)

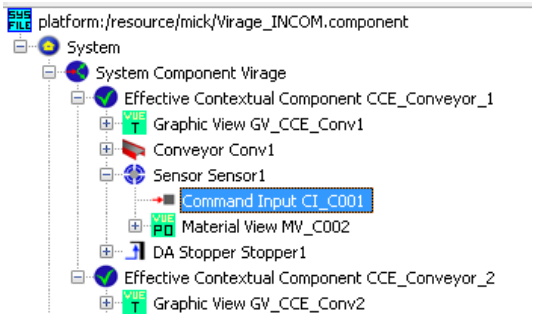

Fig. 9. Component-based model of a workstation

Fig. 10 shows the control technology model, containing all the basic components and the detail of the I/O cards.

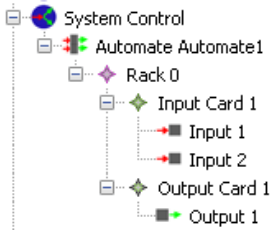

Fig. 10. Control technology model

The biggest advantage of this solution, originally intended for an industrial integrator of automated facilities, is to systematize, automate and accelerate the implementation on the customer site by the designer. Furthermore, this designer only has to be trained to model the system he implements, without any knowledge about code generation.

However, a long part of the job is made off-line, in a previous phase, by a specialist of code generation. This specialist, called modeler, has in charge to build the tools used by the designer (such as those shown Fig.10 and Fig. 11) and the model transformations to generate the codes.

The components he generates have to be generic so that they can be reused. This reusability is crucial: indeed, this approach gives better and better results each time the developed tools for one application are used again on a new implementation.

Using this approach of parallel generation of control code and observer is very helpful to establish the necessary link between the observer and the control for synchronization. Thus, the data linking problem is reduced, as the control primitives are always coded in the same way. However, the data linking is not totally solved, as the link with the DSS is not yet established.

Furthermore, another drawback relies in the genericity of the observer: as it is intended to be synchronized with the whole system state, the observer model, although customizable, remains very complex, and therefore hard to apprehend.

5.2 Designing a DSS-oriented observer

This section intends to show how the flow introduced by (Adam *et al.*, 2010) can be modified in order to cope with both the drawbacks that were shown in previous section. The

idea is to model the DSS (whose result is finally close to the content of subset $S_{needed}(t)$) in the early modeling phase, so that its features can be used in the model transformation based generation (Fig. 11).

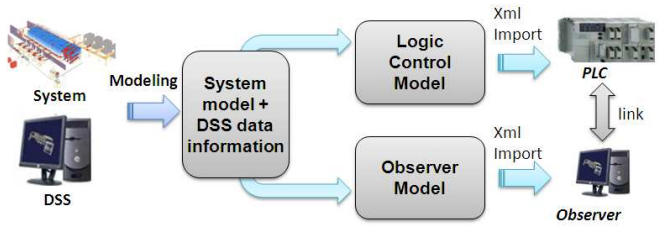

Fig. 11. DSS-oriented observer and control generation

The components defined by the modeler are instantiated by the user. Several views are attached to each component. Each view contains parameters of the system, grouped according to their characteristics (Lallican *et al.*, 2007). To include the DSS data, a new view has to be created. This view is filled with all the data of $S_{system} \cap S_{needed}$ which are needed to initialize the DSS. This set of data will be used to orient the observer, which will thus only observe the data needed and shrink.

Obviously, the content of the view being dependent of the DSS, this view has to be enriched with the description of each new DSS that is meant to be implemented.

5.3 Conclusion about the design approaches

If the second approach is globally more efficient, the difficulty for the designer is generally to model the data needed by the DSS. However, this is often relatively close to those modeling the system by itself.

The modeler deals with a greater difficulty. Indeed, the observer, being a simulation model, is classically generated using templates. The component-based generation makes the use of simulation templates coherent, as almost each component can be identified to a specific template. The definition of templates is the hardest task of the approach of the generic observer.

The case of DSS-oriented observer often requires dedicated templates, as the observation functions of each template are not compatible with the templates corresponding with another DSS. This forces the modeler to build as many template libraries as DSS modeling views. However, the templates are generally simpler, as the functions are less complex due to the component based approach.

CONCLUSION AND FUTURE WORKS

The aim of this paper is to formalize and suggest solutions for the data processing between a manufacturing system and a coupled DSS. The suggested solutions were designed for considering multiple implementations of relatively close systems.

As a matter of fact, as many procedures as possible were designed for simplicity and reusability. Of course, designing a simple solution for the designer in charge of implementation requires a higher load for the modeler to prepare the procedures.

This work is intended to be applied with an industrial integrator. Evaluation of the performance of those solutions will be performed with the actual designers and modelers in order to spot the drawbacks of these approaches.

REFERENCES

Adam, M., O. Cardin, P. Berruet P. Castagna (2011). Proposal of an Approach to Automate the Generation of a Transitic System's Observer and Decision Support using MDE. *IFAC Word Congress*

Cardin, O.(2007). *Contribution of online simulation to production activity control decision support - application to a flexible manufacture system*. Phd Thesis, Université de Nantes, Nantes

Cardin, O. P. Castagna(2009). Using online simulation in Holonic manufacturing systems. *Engineering Applications of Artificial Intelligence*, 22 (7), pp.1025-1033

Chong, C.S., A.I. Sivaknmar, R. Gay (2003). Simulation-based scheduling for dynamic discrete manufacturing. *Winter Simulation Conference*

Garavelli, A.C., M. Gorgoglione, B. Scozzi (1999). Fuzzy logic to improve the robustness of decision support systems under uncertainy. *Computer & industrial Ingenering*, 37, pp. 477-480

Lallican, J.L, P. Berruet, A. Rossi, J-L. Philippe, (2007). A component-based approach for conveying systems control design, *IFAC ICINCO*, 9 (12), pp. 329-336

Magdy, M. Abdelhameed, H. Darabi (2009). Neural network based design of fault-tolerant controllers for automated sequential manufacturing systems, *Mechatronics*, 19, pp. 705-714

Mahdavi, I., B. Shirazi (2010a). A Review of Simulation-based Intelligent Decision Support System Architecture for the Adaptive Control of Flexible Manufacturing Systems. *Journal of Intelligence*, 3 (4), pp. 201-219

Mahdavi, I., B. Shirazi , M. Solimanpur(2010b). Development of a simulation-based decision support system for controlling stochastic flexible job shop manufacturing systems. *Simulation Modelling Practice and Theory* 18, pp. 768–786

Mok, P.Y. (2009). A decision support system for the production control of a semiconductor packaging assembly line. *Expert Systems with Applications* 36, pp. 4423-4430

Pierreval H. (1992) Training a neural network by simulation for dispatching problems. *Proceedings of the Third Rensselaer International Conference on Computer Integrated Engineering*, pp. 332-336

Yang, X., H. Kumehara ,W. Zhang (2009). Back Propagation Wavelet Neural Network Based Prediction of Drill Wear from Thrust Force and Cutting Torque Signals. *Computer and information Science*, 2 (3), pp. 75-86