Emulica: an emulation-based benchmarking framework for production control experiments

Rémi Pannequin, André Thomas

Emulica: an emulation-based benchmarking framework for production control experiments [1]

Rémi Pannequin, André Thomas

Research Centre for Automatic Control (CRAN)
CNRS (UMR 7029) – Nancy University,
27, rue du merle blanc, F-88000 Épinal, France
{remi.pannequin, andre.thomas}@cran.uhp-nancy.fr

Abstract: Evaluating a proposed control architecture nearly always requires experiments and applications. This paper propose an emulation-based benchmarking framework to design and run such experiments. The proposition is based on an architecture integrating the control system with the virtual controlled system (emulation model), and on generic constructs to build this emulation model. The main feature of this architecture is presented in this paper : the constructs used to build emulation model and their generic interface with the control system. Then, Emulica, an open-source implementation of the benchmarking environment is presented.

Keywords: Emulation of complex manufacturing systems; Simulation-based performance evaluation; Software architecture

1. INTRODUCTION

Shop floor control is a key factor to be competitive. Being able to react to perturbations by taking into account the whole range of technical possibilities, while keeping the master production schedule realistic and maintaining customer service, as well as offering as-last-as-possible order modification and shorter reaction times, are some of the features promised by novel control systems. These control architectures have been the subject of many research works during the last years, and includes for instance Multi-agent control (Deen, 2003), Holonic Manufacturing systems (van Brussel et al., 1998), product-driven control (Morel et al., 2003) (Mcfarlane et al., 2003), or bio-inspired control like stigmergy (Hadeli et al., 2004) and more generally intelligent control.

A key factor in the acceptance of a novel control architecture is evaluation. Since the scientific object that is under study is as huge and complex as a manufacturing plant, experimenting on the real object is nearly always impossible, because of cost, availability and technical issues. To overcome this limitation, researchers use models of the industrial problems, either lab-sized experimental platforms, or simulation models. We have shown previously (Pannequin et al., 2009) how both approaches can be combined to form a complete experimental methodology. In the early stages of the validation of a control proposition, where the experiments are focused on the concepts and algorithms, simulation have been widely accepted as the better choice (Marik and Lazansky, 2007).

Nevertheless, using simulation in order to validate a production control system rises specific modeling needs, that the classical simulation packages might not be able to meet. While traditionally, validating a simulation model is determining if the behavior of the model correspond to reality, in the case on experiment on control systems, the question is more "does a successful experiment on a virtual (emulated) system proves that the proposed control system would be successful in reality ?", or in others word "is the experiment *honest* ?". Another key issue is the repeatability of the experiments. Indeed, such experiment are often obtained using proprietary simulation software, or custom and un-distributed simulation software.

These considerations drive us to propose in this paper a benchmarking framework that emphasize on the neutrality and independence of the emulated system with respect to control, and on the genericity of the emulation model, in order to design and to conduct experimental studies for production control research.

Fist we will develop the specific requirements for production control experiments and examine the state of the simulation software usually used. Then, section 3 will give an overview of the proposed framework, with an highlight on the generic emulation-modeling constructs and their interface with the control system. Finally, section 4 will show how this framework has been implemented in an open-source proof-of-concept software named Emulica, and section 5 will concludes.

[1] The authors gratefully acknowledge the financial support of the CPER 2007-2013 "Structuration du Pôle de Compétitivité Fibres Grand'Est" (Competitiveness Fibre Cluster), through local (Conseil Général des Vosges), regional (Région Lorraine), national (DRRT and FNADT) and European (FEDER) funds.

2. PROBLEM STATEMENT

Various kind of test-bed can be found in the litterature. For instance, comparison between antagonistic control modes such as market-based and hierarchical control (Cavalieri et al., 2000) or planning-based and reactive control (Brennan and O, 2000) have been carried out using specifically developed test beds. Others authors use Petri Net as a modelling and simulation tool (Tuncel and Alpan, 2010). Commercial simulation packages, such as Arena (Kelton et al., 2001) or Witness, are also widely used.

We can therefore see that a great variety of tools and approaches exist. However, in all of these, the integration of decision in the simulation system is also often a hard issue (van der Zee, 2006). Indeed, when using modelling paradigm based on Petri Net, or on queueing systems, it is hard to separate the modelling of the physical system (being controled) and of the control system. This very strong association between the two might enable to write easily models where the control rules are implicit (e. g. production begins as soon as a product arrives), but is a hinderance when trying to model more complex control systems.

Creating a model of the system being controled is not always a requirement: for control system that solves *one-shot* problems (e. g. off-line optimization), it can be sufficient to just make the system run on a benchmarking case (for which the optimum result is known) and see if the tested system could reach that optimum. But for systems that solves *on-going concerns* (Valckenaers et al., 2003), i.e. that dynamically reacts to events from the system being controled, simulating the physical system is the only practical option.

In this paper, we focus specifically on production control, which is an on-going concern. In this situation, there is consensus on the architecture of benchmarking environments putting the emphasis on modularity between the control system and the shop-floor system. The real shop-floor system may be replaced by a model, an *emulated* shop floor. Likewise, a model of the control system can be used instead of the real one. Therefore, four experimental situations can be defined, using either models or real systems (Pfeiffer et al., 2003) (fig. 1).

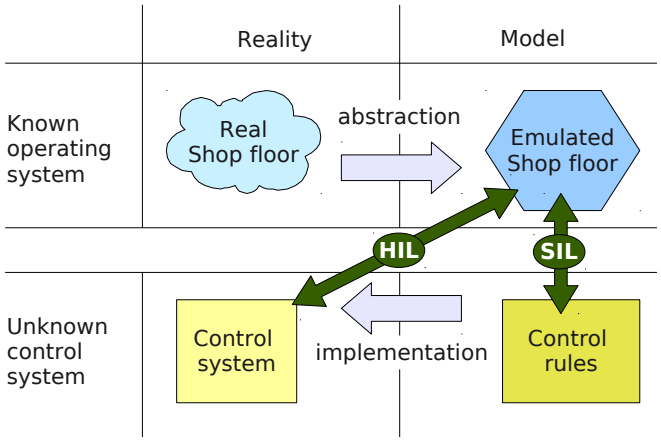

Fig. 1. The four experimental situations

The development and definition of generic benchmarking tools has drawn a great deal of interest recently, and there is a relative consensus on the architecture of such a tool. One of the key issue is to create models of complex systems (Monch, 2007), while enabling repeatability and sharing of experimental models. Other issues like performance indicators and metrics, are also very important. An on-line benchmarking utility has been defined by IMS-NoE special interest group 4 (Cavalieri et al., 2003), and (Valckenaers et al., 2006), that is based on web-based modeling of industrial benchmarking cases, that could then be shared to the community. This approach seems very promising, but put all the pressure on a central server. A common format for emulation models (and therefore generic emulation constructs) might also be a solution to this issue.

Nevertheless, modeling issues remain : what kind of constructs are necessary and sufficient to create emulation models ? What kind of interface with these constructs should exists ? Where should lies the boundary between control and emulation systems ?

3. ARCHITECTURE FOR AN EMULATION-BASED BENCHMARKING TOOL

3.1 Modeling Principles

Three main use cases can be distinguished for a benchmarking tool for production control research. At the first stages of the development work, it must enables researchers to dynamically play with the test system, and therefore gain knowledge of the issues that might not be visible in a static description. Then, the benchmarking tool must be able to validate the proposition by comparing the proposed control approach and the state-of-the-art approach on one or several test cases. Finally, the benchmarking tool should enable other researchers to reproduce the experiments and analyze them.

From these three basic uses cases, several properties that a control-oriented benchmarking tool should have can be deducted. The first one is the ability to represent the problems that the researcher try to solve, and any other aspect of the reality that might have an impact on the experimental results. The second one is a strong dissociation between the control system being tested, and the virtual shop floor system being controlled. In order to compare various control strategies, test case must be entirely control-neutral. Finally, the third requirement are generic modeling components.

In order to achieve this requirements of genericness and expressiveness, our proposed emulation framework is grounded on widely accepted principles.

The first one is to focus the modeling on products, and on their various transformation processes. The emulator have to simulate the dynamics of the physical transformations of the products. The state of a product is characterized by its spacial, morphological and temporal components. So, the product will evolve in space and the set of its possible shape, through time. The emulator therefore includes shape and space transformation processes (fig 2). The products can also be combined together to form assemblies or be disassembled. Finally, the emulation must include

creation (i.e. entry in the model) and destruction (i.e. exit of the model) processes.

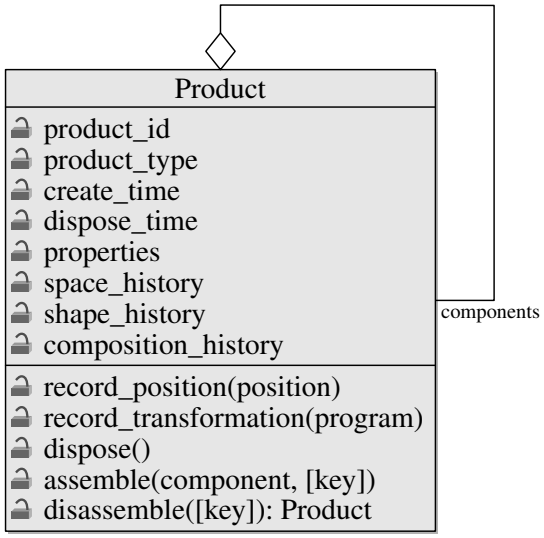

Fig. 2. Model of emulated products, characterized by their spacial and morphological attributes

The transformations of products are not free, and the goal of the emulator is to represent the *physical constraints* on the spacial and morphological trajectories of the products. This includes, on the one hand, the dynamic of the physical processes (e. g. shared resources, capacity constraints, speeds) and, on the other hand, the input of the control system.

The concept of the cybernetic loop, enables us to distinguish three kind of constructs (fig. 3):

Actuators that change the products spacial, morphological or structural attributes when receiving requests from the control system,
Processes that model the uncontrollable constrains on products (e. g. queue capacity) and on other modules (e. g. failures),
Observers that observe the products and send reports to the control system.

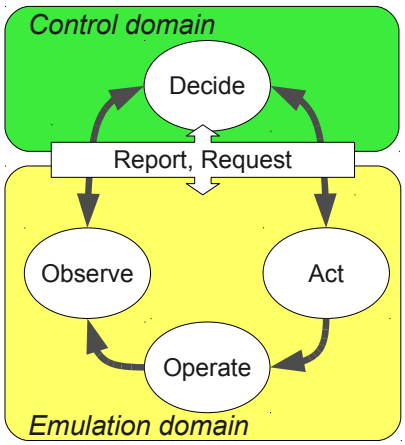

Fig. 3. The cybernetic loop, where the domain of emulation and control are separated

Moreover, actuator not only receive request, but also report to the control system when their state change. likewise, observers can be triggered directly by the control system (i.e. by receiving observation requests).

To ensure that the messages exchanged between the emulator and the control system follow a common structure, while keeping them simple and generic, the rule of the six Ws has been used. Reports and Request are completely determined by the six attributes:

who the module that send (resp. receive) the report (resp. the request)
what nature of the state to report (resp. of the action to execute)
when date of of occurrence (resp. of execution)
how parameters of the report (resp. of execution)
where place of occurrence (resp. of execution)
why human interpretation : a commentary

3.2 Emulation constructs

The main constructs that are used in the emulator are the Model, Products and Modules. A model is made of modules and products. Additionally, a Model is a kind of Module (by using the composite design pattern), which enable to include (sub)models in models, thus providing versatility : custom "macro-modules" can be built to suit a specific case, and provide practitioners with re-usable high level modeling blocks.

The various kind of actuators, processes and observers are all subtype of Module. Each module has a standard structure (fig. 4) :

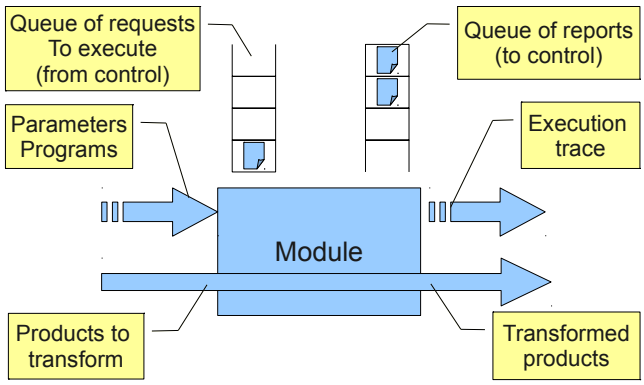

Fig. 4. The structure of a module

The most basic module is the Holder : its role is to contain one or several products. Holders can have a limited or infinite capacity. Holders have only one entry and one exit point, which make possible to use them to designate a position where to fetch or put products. Inside the Holder, products flows from the entry to the exit following a queue discipline (FIFO or LIFO).

Observers are connected to Holders, and either send a report to the control when a product is available in the Holder, or scan the entire holder for products when requested. In the first case, the information is pushed to the control system (PushObserver) and in the second one,

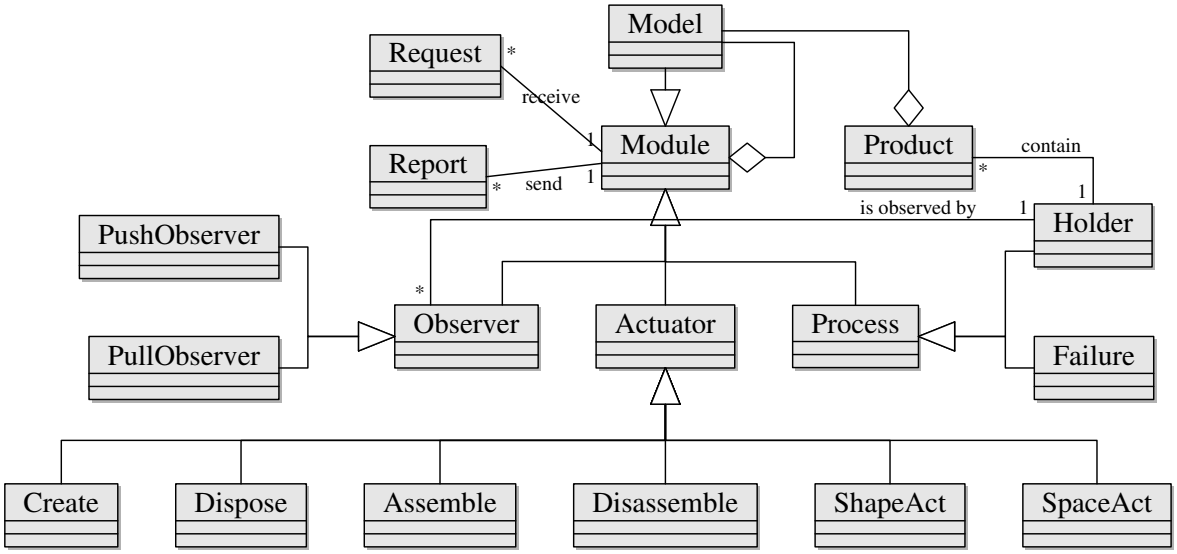

Fig. 5. Simplified Class diagram of the main emulation elements: Each kind of module is represented, as well as Products, reports and requests

the control system pull the observation events (PullObservers).

There is six type of actuators: CreateAct and DisposeAct, ShapeAct and SpaceAct, AssembleAct and DisassembleAct. All the actuators operate on products, and are therefore linked to one main holder, that represents the space where the product being worked on resides. Each actuator execute programs. A program encapsulate a basic action that can be performed by the actuator. It is identified by its name, and is characterized by its execution time and by the physical transformation that have to be applied on products. For a Shape actuator, a program change the properties of the products. For a Space actuator, a program moves a product from a source holder to a destination holder. For an Assemble actuator, the program fetch a product form a source holder, and assemble it as a component of the product currently loaded in its main holder.

An actuator can execute a set of programs, which are stored in a program table. One of the programs is the current program, and can be run at once. To change the current program, a setup must be done. Each setup is characterized by the initial and final programs, and by the setup delay.

Products and Modules can have properties, identified by there names, which represent their physical characteristics. Theses properties can be expressions: for instance the mass of an assembly is the sum of the mass of all the components.

In programs and setups, the delays can be constant time, or evaluable expressions. When evaluating an expression, special token can be used: *rng*, the model's random number generator (for stochastic time law), the current product (and its properties) and the current module (and its properties). That way, it is possible to model transformation times that depends of the characteristics of the product being transformed (e. g. its mass, or length), or of a characteristic of the module.

3.3 Control integration

As presented in figure 1, There is two different type of experiments that could be done using the emulator: On the one hand, the real control system can be used, or, on the other hand, only a model of the control system can be used. In the latter case, the control system is a set of control functions and data structures, that are executed inside the emulator's simulation environment. In the first case, the emulator wait for the external control system to connect on an interface (e. g. a network socket) where it will get requests and send reports.

In both control integration mode, the interaction between control and emulation is done uniquely through reports and requests. A typical control process will wait reports from the observation modules, change its internal data structures and send requests to the emulated shop floor.

This ability to nest the control system in the emulator is useful to easily test control algorithms without the burden of developing a full control software. In this integration mode, the control processes are discrete-events routines just like the emulation modules. Both share the same event scheduler, making possible to run discrete-events simulation, (i.e. the time jumps to the next event).

However, when the control system is external to the emulator (and have therefore no access to the event scheduler), a real-time simulation is more appropriate (i.e. the advance of time in the simulation is proportional to the computer clock). It is also possible to use hybrid time advance (Saint Germain et al., 2003), where the emulation runs in real-time when the control system is active, and otherwise jumps to the next event.

The last issue in control integration is how to include submodels. As stated in the diagram in figure 5, Models can integrate submodels (i. e. a Model is a sort of Module). Submodels comprise an emulated physical system, and their own (nested) control system: a submodel is a complex discrete-event process made of several emulation and control discrete-event processes. Using this feature,

it is possible to create complex constructs, for instance building block for a particular type of application.

For instance, a commonly used object in emulation model is a manufacturing cell. A simple manufacturing cell may be modeled with three products containers (input and output buffers, and the space inside the machine), a space actuator modeling the loading / unloading functions, a shape actuator modeling the transformation process, and at least one observer that monitor the products in the input buffer. Some control functions may be developed in order to control this cell, like for instance to follow a given production plan (execute program p_1 on n_1 products, then p_2 on n_2, etc).

Therefore, to be re-used in another emulation model, this simple cell model is able to receive parameters (like for instance a table of the program that this particular cell is able to make, or loading / unloading times, etc) at modeling time, as well as taking requests as inputs (with, as in this example a production plan), and sending back reports at run-time. So, two distinct interfaces should be defined when designing a submodel : a control interface, where the control messages will be exchanged, and a configuration interface, which enable to use several instance of the same submodel with different execution parameters.

4. EMULICA : AN IMPLEMENTATION OF THE PROPOSED ARCHITECTURE

Emulica is a software that implements the emulation architecture described above. This implementation has two goals:

- validate and demonstrate the proposed architecture,
- create a re-usable sofware tool to suport research.

It is an open-source project. This enable to reuse as much as possible open source components: SimPy provides the discrete-event simulation core, while (py)GTK, (py)goocanvas, (py)gtksourceview and matplotlib are used to build the graphical interface, display the emulation models, edit the control system code and create results plots. Being open-source also mean to user they can easily modify the tool to their own needs.

Emulica a set of python modules that can be used in external applications or scripts, and a graphical user interface. The interface is divided in three parts:

- graphical editing and animation of emulation models,
- control editor with syntax coloration and code snippets
- graphical result generation and visualization

The emulation modeling and execution is implemented on top of the SimPy discrete events simulation package. Each kind of emulation module has a *process execution method* that call the relatively low-level SimPy API. Model edition can be done visually in the graphical user interface (fig. 6), or programatically using emulica API. Thanks to this double approach, novice users can get started with emulica's concepts, while advanced use-cases like embedding the emulator in another program or automatic model generation are possible.

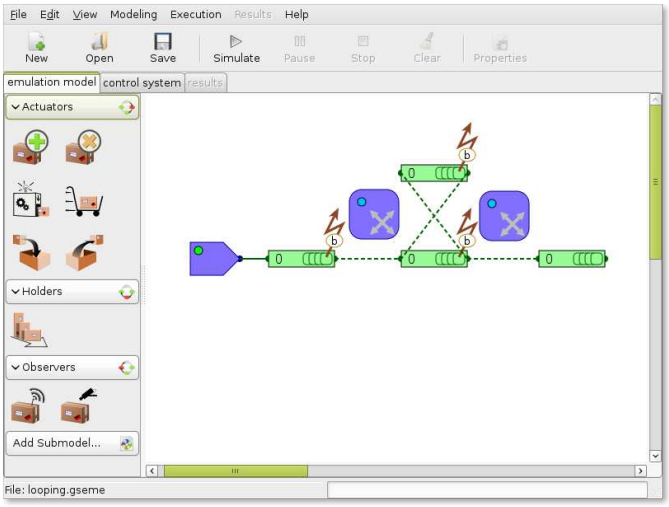

Fig. 6. Edition of an emulation model

The control edition part (fig. 7) take advantage of the features of python. First, python's generators are used to write the control processes. A generator can be described as a function that can return several values, and that stays in its state after returning a value. The control process will return statements meaning "to get an report from a module" (or wait until it arrives), and "to send a request to a module". Thanks to generators, such statements can be sequentially issued, like in a sequential flow chart. Then, control edition take advantage of the very powerfull reflexivity features of python : emulica can compile the user-entrered code of the control system, insert it into the model and run it.

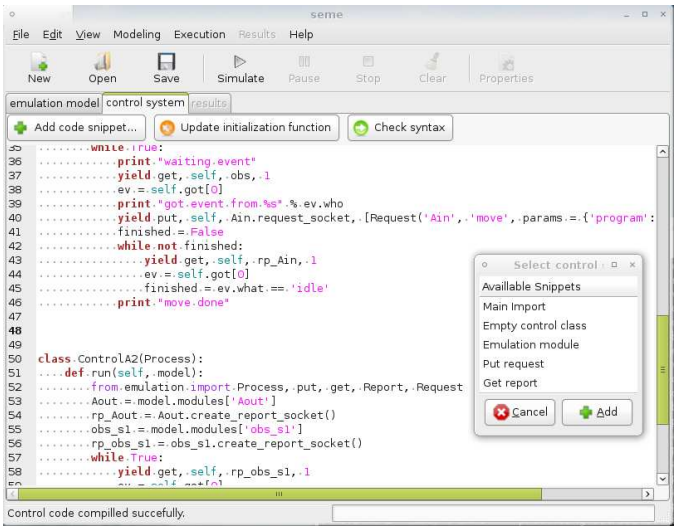

Fig. 7. Edition of a nested control system

Emulica is still in early development and is beta-quality software. The software can be downloaded freely, with its source code, examples and documentation, from the web at http://emulica.sourceforge.net. It is available for Windows and Linux, and should be portable to any platform where python is supported.

The validation of Emulica's implementation of the emulation constructs is done using a set of test-cases, where the theoretical execution trace is compared with the one

obtained with the emulator. Emulica is also currently used in several research projects, with coupling between emulica and an agent platform (JADE), or coupling between Emulica and a linear programming solver (glptk).

5. CONCLUSION

We have shown in this paper that a set of around ten modeling primitives, used in conjunction with the composite pattern, may be able to model many type of systems. Another preliminary results is a way to delimit emulation models and control systems, is the case of composite modeling : the control system and the emulation model are merged together to form a new emulation component. Thus, the strong independence between control and emulation model, is only valid for one level of detail, which corresponds to the control problem being studied.

An emulation tool based on this original modeling approach have been introduced, specifically oriented toward production control experiments. Being open-source, its architecture, inner behavior, and code may be easily reviewed (and modified). This is in our sense the right way to create and share a re-usable tool.

At the time of writing, the proposed emulation framework is more a proof-of-concept for the emulation constructs and architecture. To become a real benchmarking system, the framework will have to be faced with users expectations, and modified accordingly. It seems to us that this process should be driven by the users themselves. Nevertheless, it is already in an usable state, and, as the number of application grows, we will be able to have a better validation of the genericity of the emulation constructs, and add –if required– some new features.

We are aware that this paper only briefly deals with a subject that requires more in-depth analysis. Our goal was to give an overview of the core architecture of the proposed benchmarking system, and highlight the issues of generic emulation components. As perspectives, we plan to present with more detail the various aspects of the benchmarking framework in future papers.

REFERENCES

Robert W. Brennan and William. O. A simulation testbed to evaluate multi-agent control of manufacturing systems. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 1747–1756, San Diego, CA, USA, 2000. Society for Computer Simulation International.

S Cavalieri, M Macchi, and Valckenaers P. Benchmarking the performance of manufacturing control system: design principles for a web based simulated testbed. *Journal of Intelligent Manufacturing*, 14(1):43–58, 2003. doi: 10.1023/A:1022287212706.

Sergio Cavalieri, Marco Garetti, Marco Macchi, and Marco Taisch. An experimental benchmarking of two multi-agent architectures for production scheduling and control. *Computers in Industry*, 43(2):139–152, October 2000. doi: 10.1016/S0166-3615(00)00063-4.

S. M. Deen. *Agent Based Manufacturing*. Springer, July 2003.

Hadeli, Paul Valckenaers, Martin Kollingbaum, and Hendrik Van Brussel. Multi-agent coordination and control using stigmergy. *Computers in Industry*, 53(1):75–96, January 2004. doi: 10.1016/S0166-3615(03)00123-4.

D. Kelton, D. A. Sadowski, and R.P. Sadowski. *Simulation with Arena*. McGraw-Hill, 2001.

Vladimir Marik and Jiri Lazansky. Industrial applications of agent technologies. *Control Engineering Practice*, 15(11):1364–1380, 2007. doi: 10.1016/j.conengprac.2006.10.001.

Duncan Mcfarlane, Sanjay Sarma, Jin L. Chirn, C. Y. Wong, and Kevin Ashton. Auto id systems and intelligent manufacturing control. *Engineering Applications of Artificial Intelligence*, 16(4):365–376, June 2003. doi: 10.1016/S0952-1976(03)00077-0.

Lars Monch. Simulation-based benchmarking of production control schemes for complex manufacturing systems. *Control Engineering Practice*, 15(11):1381–1393, 2007. doi: 10.1016/j.conengprac.2006.05.010.

Gerard Morel, Herve Panetto, Marek Zaremba, and Frederique Mayer. Manufacturing enterprise control and management system engineering: paradigms and open issues. *Annual Reviews in Control*, 27(2):199–209, 2003. doi: 10.1016/j.arcontrol.2003.09.003.

Rémi Pannequin, André Thomas, and Gérard Morel. Proposition d'un environnement d'évaluation pour la mise en oeuvre d'un pilotage par le produit. *Journal Européen des Systèmes Automatisés*, 43(4-5):487–511, 2009. doi: 10.3166/jesa.43.487-511.

A. Pfeiffer, B. Kádár, and L. Monostori. Evaluating and improving production control systems by using emulation. In *Applied Simulation and Modelling*, 2003.

B. Saint Germain, P. Valckenaers, C. Zamfirescu, O. Bochmann, and H. Vanbrussel. Benchmarking of manufacturing control systems in simulation. In *Proc. of the 3rd Int'l Workshop on Performance Measurement (IfIP WG5.7), Bergamo*, pages 357–369, 2003.

Gonca Tuncel and Gülgün Alpan. Risk assessment and management for supply chain networks: A case study. *Computers in Industry*, 61(3):250 – 259, 2010. doi: DOI: 10.1016/j.compind.2009.09.008.

P. Valckenaers, Hendrik van Brussel, H. Karuna, O. Bochmann, B. Saint Germain, and C. Zamfirescu. On the design of emergent systems: an investigation of integration and interoperability issues. *Engineering Applications of Artificial Intelligence*, 16:377–393, 2003.

Paul Valckenaers, Sergio Cavalieri, Bart Germain, Paul Verstraete, Hadeli, Romeo Bandinelli, Sergio Terzi, and Hendrik Brussel. A benchmarking service for the manufacturing control research community. *Journal of Intelligent Manufacturing*, 17(6):667–679, December 2006. doi: 10.1007/s10845-006-0036-y.

Hendrik van Brussel, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37(3):255–274, December 1998. doi: 10.1016/S0166-3615(98)00102-X.

D. J. van der Zee. Modeling decision making and control in manufacturing simulation. *International Journal of Production Economics*, 100(1):155–167, March 2006. doi: 10.1016/j.ijpe.2004.11.001.