Automatic generation of simulation code using product location information

Andres Véjar, Patrick Charpentier

Automatic generation of simulation code using product location information

A. Véjar P. Charpentier

Research Centre for Automatic Control (CRAN), CNRS UMR 7039, University of Nancy, BP 70239, Vandœuvre-lès-Nancy 54506, France (e-mail: {andres.vejar,patrick.charpentier}@cran.uhp-nancy.fr)

Abstract: This article proposes an original method for simulation code generation in discrete event systems. This method uses the product location information in the running system. The information flux (product *id*, location, time) is the starting point for the algorithm to generate a queuing network simulation model.

Keywords: Flux, Location, Product, Simulation, Modeling.

1. INTRODUCTION

Few years ago, a new perspective for product design and manufacturing incorporated into the product, communication and sensitive capabilities within the framework of the intelligent product paradigm, (Karkkainen et al., 2003).

The informational part of each product is feeded by the direct material environment of the physical product or by its own instrumentation (MEMS,GPS, ...) The exchanges of information between the product and its environment can be made:

- At certain synchronization points using RFID technologies, bar code...
- In a quasi-continuous way (wireless networks like Wifi, Zigbee, Bluetooth)

These communication technologies can also contribute to the product location activity in addition to the embedded instrumentation. There are many applications of these technologies in the field of production and logistics, some examples are products traceability, stock inventory and the positioning of a transport fleet. By now the spatial location of physical objects, is limited to voluminous objects (lorries, boats, containers) or to people.

The aim of our research consists in showing the new inputs and benefits, in the use of a product location information flux, during the manufacturing process. This article presents an application: the automatic generation of flux simulation code on the basis of product location data during its passage through the production system. This data compose an information flux that can be assimilated as product traces.

If since some years, the simulation of product flux has been the main tool for the evaluation of the dynamic of manufacturing systems, (Cassandras and Lafortune, 1999; Park and Lee, 2005), it has often been shown that the modeling phase and the maintenance phase are constituted by delicate and time- demanding human operations (De Vin et al., 2004; Mittal et al., 2005; De Vin et al., 2006). These reasons explain in itself the choice of this problem. The main idea is to replace the biggest amount of human expert interventions in the construction of the simulation model, but at the same time in the maintenance and reconfiguration phases, for an automatic generator.

The figure 1 shows schematically the principle. The generator is feeded by a data flux that comes from the real system.

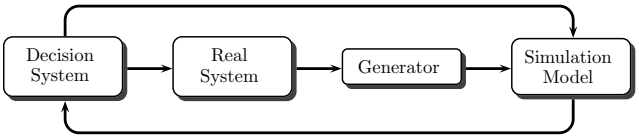

Fig. 1. The generator and its environment

2. HYPOTHESES & PROBLEM STATEMENT

2.1 Initial Hypothesis

The main hypothesis of this work is to consider that all of the elementary objects of a manufacturing system can be located. Suppose also that exist a technology capable of providing a location flux of all of this moving objects in the system. We talk in this case of location rather than geolocation, because of the nature and the scale of the study system. The scale is limited by the workshop size. Another hypotheses can be naturally added to complete the main hypothesis. In effect, the observed data is considered in this work as reliable and without error, our idea is to show the principles of the proposed method in the most simple manner (the consideration of noisy data will be the object of other works and presentations). Moreover, the location data acquisition can be made naturally through sensors either embbeded in the product or not, that through a communication system can feed an information management system (Figure 2).

The location data are, as others physical quantities, a result from the environment of product circulation, the manufacturing system. The access to location data can be

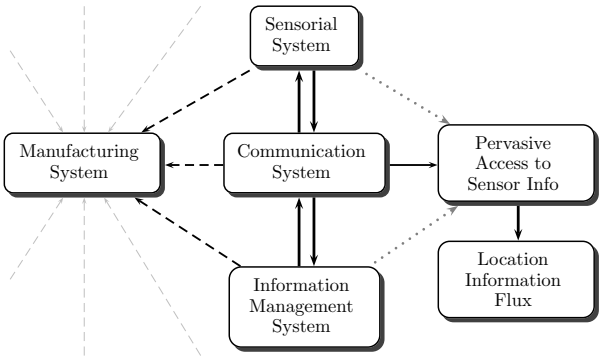

Fig. 2. To get the location flow...

possible in all the positions in the system in each instant, for this, the system must be a pervasive one.

In our case we can imagine that the data is collected in each event, or with a discrete sampling time (depending on the technology used). In this last case, with a high frequency of data acquisition, the location information flux can be regarded as quasi-continuous. To finish, we consider that a single, globally unique *id* is assigned *a priori* for each elementary product-object handled in the production system.

2.2 Problem Formalization

The accessible location information is defined by the 3-tuple (I, R, T) where I is the set of *Ids* for objects. Only one unique *id* is assigned for each object. The set $R \subset \mathbb{R}^2$ is the set of positions ($r = (x, y)$, $r \in R$), and T represents the time. Each (i, r, t) is an element in the flux f. This is all the information that will be used for the generation of simulation code. Our problem consist in conceiving a generator of simulation models on-line Φ, capable to develop a model m from the real data flux F.

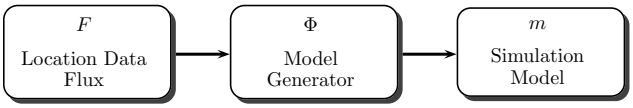

Fig. 3. On-line generator

The generator, Φ, must be capable of adapting m in function of the evolution of flux F in time (2). In a more formal manner we can write:

$$F = \{f_1, f_2, f_3, \ldots\}, \tag{1}$$

where f_k represent the flux at a given time, the it is a component of the flux F. We considered that the model was empty at the beginning. The arrival of a new flux to the generator allows the update of the model m. The problem consist on defining Φ in order to obtain the model m for each change in the flux. This procedure allows to adapt the model between modifications of the real system: the generation can react to reflect the changes.

The recursive map of the procedure to obtain the model m is presented (with m_0 the initial empty model):

$$\begin{array}{rcl} m_0 & = & \emptyset \\ m_k & = & \Phi(f_k, m_{k-1}) \end{array} \tag{2}$$

3. MODELING

3.1 Objects and products

In this work we consider the product as an object (or objects) under development, since its birth in the entry of the system, until his disappearance to his release. Objects, composited or not, can be assembled: the outcome of the assembly is a conglomerate of objects named product. A product can be disassembled: this process generate the most basic objects. These are the most basic objects that are linked to a unique identifier.

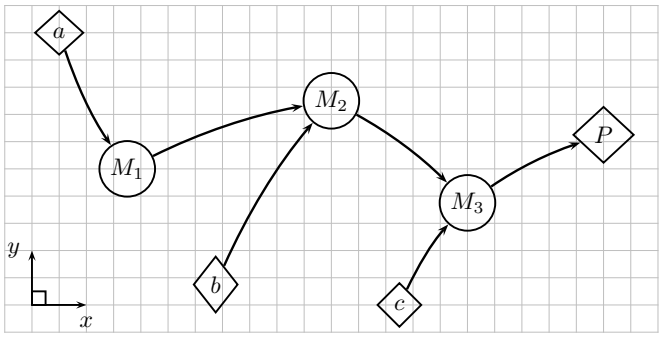

Fig. 4. Workshop, three machines, one product

This notion of objects and products will be introduced on the basis of a simple example of a workshop with three machines M_1, M_2 et M_3 and one type of product P (4) this notion of objects and product. In this example, the product is composed of three objects a, b, c. The product P is the gradual agglomeration of initial objects (a, b, c) in intermediate objects (p_1, p_2) (6). It may represent these processes formally:

$$\begin{array}{rcl} a & \mapsto & M_1(a) = p_1 \\ (p_1, b) & \mapsto & M_2(p_1, b) = p_2 \\ (p_2, c) & \mapsto & M_3(p_2, c) = P, \end{array} \tag{3}$$

or more synthetically for a composition of functions:

$$P = M_3\left(M_2\left(M_1(a), b\right), c\right) \tag{4}$$

We can represent the equations (3) and (4), as a graph $G = (V, E)$ (4), with vertices:

$$V = \{a, b, c, M_1, M_2, M_3, P\} \tag{5}$$

and the adjacency matrix is:

$$Adj(V) = \begin{pmatrix} 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0 \end{pmatrix} \tag{6}$$

This last formalization, in numeric form, will be useful during the implementation of our algorithm to determine the composition of final products.

3.2 Trajectory and history of products

This part of the research takes the example introduced in the previous paragraph. On Figure (4) are represented two different things:

- Composition (or nomenclature) of a type of product.
- Location in the workshop (cartesian coordinates x and y) resources needed to develop P and the path of objects, whether elementary or composite.

On this type of scheme the diamonds represent the birth or the disappearance of objects and/or products (a diamond with an arrow outgoing is a point of birth, a diamond with an arrow entering is a point of disappearance) considered for the study.

The monitoring of the trajectory of an object in the plane (x, y) in time allows us to determine the speed v: either it is zero, or it is positive (5) The product is moving or is stopped: the latter property is a sign of a waiting time... hence the choice to build a simulation model based on a queueing network. The information $v = 0$ located a particular point at the base of the construction of our model. Because of each of these points it is possible to obtain an histogram that represent the service time for each product type. However, before the disappearance of a product, no information on its composition is available. The objects (initials and intermediaries) provide no information about their final destination. It is in the disappearance of the final product where the composition is discovered (by the joint movement of its components 6). It is then possible to trace all the pathways of its constituents to the moment of their birth, and update the histograms of sevice time of a part for a final product on a point $v = 0$.

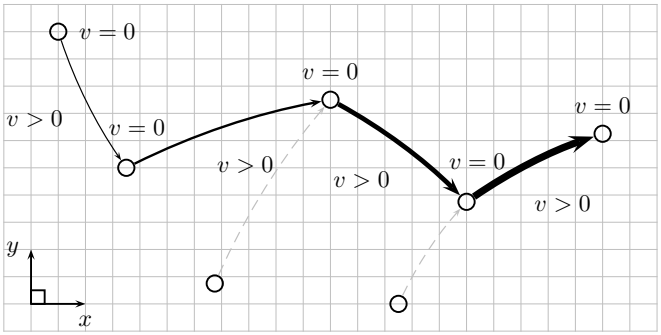

Fig. 5. Velocity component a

3.3 Waiting Queues

At each point where $v = 0$, we associate a waiting queue behavior (7).

In these points two types of events may occur: the arrival or the departure of object(s). As the composition of a product can be only known in the output of this waiting queue, the objects constituents of the product then follow the same trajectory. Then we define (8):

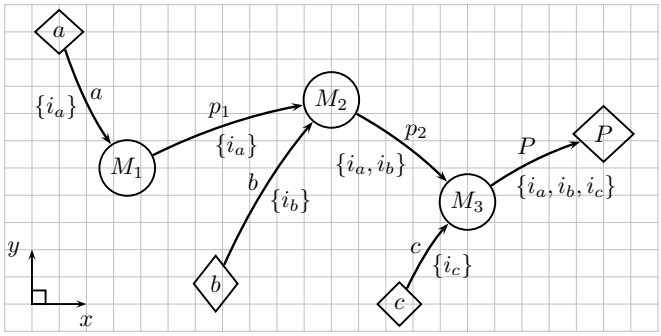

Fig. 6. Workshop, indicators and objects flow

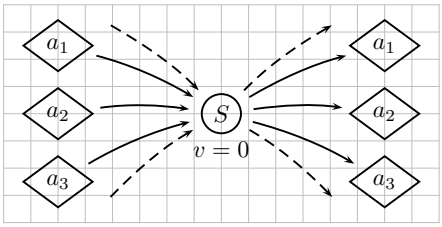

Fig. 7. Service point, $v = 0$

- e_1 the instant of arrival of the first object that constitutes the product.
- e_2 the instant of arrival of the last object that constitutes the product.
- s is the common instant of departure all objects that are constituents of the product.

On this basis we can get:

- O, the time between the arrival of the first object and the arrival of the last object needed for the product constitution.
- A, waiting time for service : defined as the period between the time when all the pieces needed have arrived, and the instant when the service start.
- T, service time at point $v = 0$.
- T_{Tot}, total waiting time at point $v = 0$.

We can link these different variables in the equation:

$$T_{Tot} = O + A + T \tag{7}$$

To compute A is necessary to know the departure instant of the previous product s^{k-1} where $k - 1$ the order of departure of the products. The rule for calculating A is: if $e_2^k < s^{k-1}$ then the waiting time is $s^{k-1} - e_2^k$, else the waiting time is 0.

In a more formal manner and using of the Heaviside function:

$$\theta(y) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0, \end{cases} \tag{8}$$

now we can define at each k product departure, the variables introduced so far:

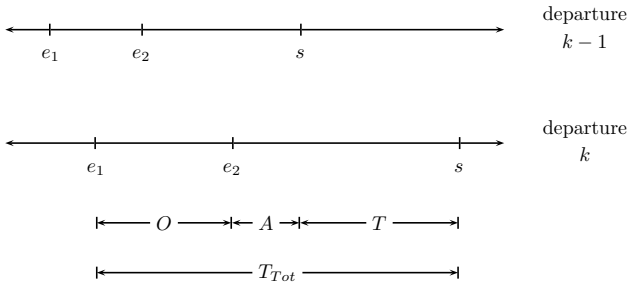

Fig. 8. Waiting periods at congestion point $v = 0$

$$\begin{aligned} y^k &= e_2^k - s^{k-1} \\ \alpha^k &= \theta(y^k) \\ \beta^k &= 1 - \alpha^k \\ T^k &= \alpha^k(s^k - e_2^k) + \beta^k(s^k - s^{k-1}) \\ A^k &= \beta^k(s^{k-1} - e_2^k) \\ O^k &= e_2^k - e_1^k \end{aligned} \tag{9}$$

with $s^0 = 0$.

The 3-tuple (T, A, O) has all the necessary information in order to characterize the point $v = 0$ as a service point.

We must note that the analysis presented here is valid only under the theoretical condition that the queue is reduced to one point (all objects waiting at the same point).

4. ALGORITHM

The algorithm proposed here is naturally the generator of the simulation code. It comprises three main parts. The first part generates the machine positions and the paths followed by products, on the basis of the real-time of data flow. The second part deals with the issue of products departures of the system. The product departure is the instant when is possible to know whether it type existed before or not. If the type existed, it updates the product data, otherwise a new product type is created. The third part is used to modeling the laws of behavior for the simulation model. Statistical laws are proposed to represent the inter-arrival time and service time for each product type at each machine.

Table 1. Data structure m

var	description	defaut
$m.r$	position (x, y)	none
$m.t$	current time	none
$m.s$	binary for stop condition	0
$m.T$	service time	0.0
$m.l$	departure instant	0.0
$m.a$	stop time	0.0

The algorithm associated with this first part is triggered with every change in the flow. At each new data (i, r, t) a new product is created if the data *id* is not know. At each stop of this product it is created a *stop point* (location point of a queue and/or service point), and a link between product and the *stop point*. This *stop point* is validated if the position of the product i is in the same position between the instants t and ts, $t < ts$, then the point corresponds to a real waiting point. In this case the time of arrival and departure of the product i on the waiting point located in r are retained. They are used for modeling the dynamic behavior of this type of product at this point.

The algorithm associated with this second part is triggered at each product departure of the system. Whether a product outcomes of the system may be a difficult problem to solve. We have chosen to define a product as out of the system if the identifier i no longer appears in the data flow during a considerable period (chosen arbitrarily but large enough). As we have already explained in the equation (3) or in figure (4) it must be only possible to know the type of product at the exit. In this level we use the adjacency matrix linked to each elementary product. The sum of these matrices to each elementary product that departs out of the system at the same space-time instant can return the composition of the final product. The algorithm compares this sum of matrices with those obtained previously. If this matrix does not exist, it means that a new product came out of the system. Then it connects the information on its service (or stop) points obtained during the movements of its various components in the workshop.

The third part is made to characterize the stochastic behavior of different stop points. This third and final phase is the phase of post-treatment of all the information collected in phase 1 and 2. An estimation of the probability density function for the inter arrival time and service time is driven using kernel density estimators. It is then validated by a Wilcoxon test (Wilcoxon and Co, 1997).

We present (1) a fragment of the algorithm developed in its simplified version. This fragment can collect the positions of stop points for each product, and their inter-arrival time and service time at each stop point.

In the presented algorithm a point of data (i, r, t) is defined by the structure d, where $d.i$ is the *id*, $d.r$ the (x, y) position, and $d.t$ the time.

Other data structures used are m and p. The structure m is defined in the table (1), and the function to create a new structure is $\breve{m}(r, t)$.

The structure p is defined in the table (2), and the function to create a new structure is $\breve{p}(i, m)$.

Table 2. Data structure p

var	description	defaut
$p.i$	*id*	none
$p.m$	structure of type m	none
$p.M$	ordered set of m structures	$\emptyset$

In addition to the structures d, m, p , there are two global sets M and P, empties by default.

5. APPLICATION

This section is intended to validate the principle of feasibility of the automatic simulation code generation on the basis of location information.

In first time we generated a simulation module using Simpy (Simulation in Python) the object-oriented, process-based discrete-event simulation language (Muller, 2004; Muller and Vignaux, 2003; Bahouth et al., 2007). In its module we generated the product location information

Algorithm 1 Simplified algorithm

for d **do**
  if $\exists\, p \in P\colon p.i = d.i$ **then**
    if $p.m.s = 1$ **then**
      if $p.m.r = d.r$ **then**
        $p.m.a \leftarrow p.m.a + d.t - p.m.t$
        $p.m.t \leftarrow d.t$
      else
        $m \leftarrow m' \in M\colon m'.r = p.m.r$
        if $p.m.t - p.m.a < m.l$ **then**
          $p.m.T \leftarrow p.m.t - m.l$
        else
          $p.m.T \leftarrow p.m.a$
        end if
        $m.l \leftarrow p.m.t$
        $p.M \leftarrow p.M \cup \{p.m\}$
        $p.m \leftarrow \breve{m}(d.r, d.t)$
      end if
    else
      if $p.m.r = d.r$ **then**
        $p.m.s \leftarrow 1$
        if $\nexists\, m \in M\colon m.r = d.r$ **then**
          $M \leftarrow M \cup \{p.m\}$
        end if
      else
        $p.m.r \leftarrow d.r$
      end if
      $p.m.t \leftarrow d.t$
    end if
  else
    $m \leftarrow \breve{m}(d.r, d.t)$
    $p \leftarrow \breve{p}(d.i, m)$
    $P \leftarrow P \cup \{p\}$
  end if
end for

flux. It acts as an artifact of the real system. We limited our research here voluntarily to a simple case to limit the volume of its presentation and analysis of results. The location flow is then retrieved by the generator module of simulation model which generates simulation model implementing phases 1 and 2 described before. Finally, a module for data analysis (phase 3) allows to verify the results. All the modules are programmed in Python (Rossum, 1995; Downey et al., 2002; Cai, 2005).

5.1 Presentation of the chosen application

The workshop modeled here is composed of different machines among which the products are evolving, transported by AGV's.

The different types of products are generated for the formula:

$$mach(s, l) = (s - l) \mod M \tag{10}$$

- l: product type, $l = 0, \ldots, L-1$,
- s: operations, $s = 0, \ldots, S-1$,
- M :quantity of machines,

based on the original formulation proposed by Thiesse and Fleisch (2008), The proposed formula is used to generate sequences of processing for each product type.

- The inter arrival time of products in the workshop are provided by an exponential distribution. Each product type is generated in the same proportion.
- The service time of each machine is also provided by an exponential distribution function.
- The machine positions are carried out randomly between the dimensions of the system at the beginning of the simulation.
- The tracking system allows to obtain location information Product (i, r, t) with a fixed frequency.

5.2 Implementation with SimPy

The parameters used in the simulation are:

Table 3. Experimental conditions

parameter	value
product types	3
machines	3
operation for each product	3
products	100
inter arrival time	exponential, $\mu = 1/3$
service time	exponential, $\mu = 13$
AGV's	10
AGV's velocity	0.44
workshop limits	$(-10, -10), (10, 10)$
location sampling period	0.5

5.3 Results

The results provided by the model from the code generator are quite consistent with those obtained from the artifact of the real system. Indeed, the Wilcoxon test, shows that data on inter-arrival time and service time generated by the two models are distributed under the same statistical law. (Tables 4 et 5). The products and their types are fully identified by the code generator. It also sets out in detail the differents resources implemented for the development of products. These early results show the feasibility of the proposed method. They were confirmed by tests on larger

Table 4. Wilcoxon test, inter arrival time

Product	Operation		
0	0	1	2
t-statistic	168.5	138.0	131.0
two-tailed p-value	0.7	0.0	0.0
1	0	1	2
t-statistic	96.0	5.0	124.0
two-tailed p-value	0.49	0.0	0.0
2	0	1	2
t-statistic	240.5	195.5	192.5
two-tailed p-value	0.73	0.0	0.0

Table 5. Wilcoxon Test, service time

Product	Operation		
0	0	1	2
t-statistic	220.0	187.0	185.5
two-tailed p-value	0.8	0.35	0.3
1	0	1	2
t-statistic	154.0	175.0	166.0
two-tailed p-value	0.07	0.15	0.17
2	0	1	2
t-statistic	261.0	270.0	309.0
two-tailed p-value	0.07	0.09	0.37

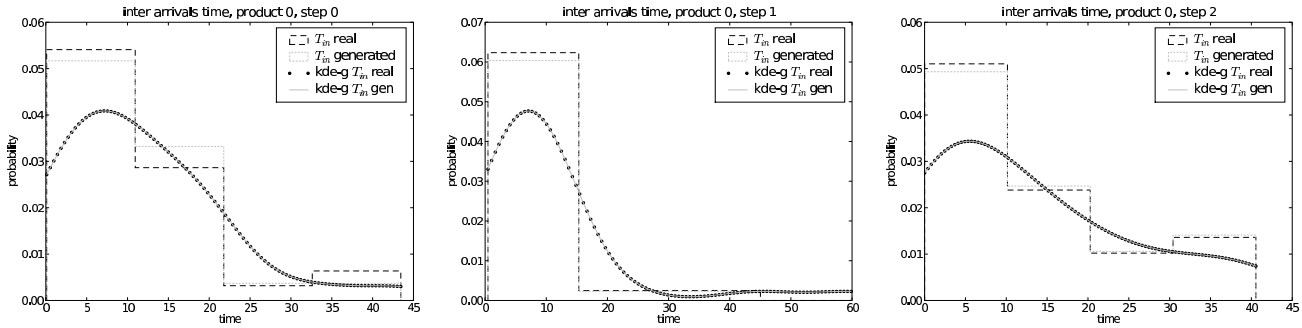

Fig. 9. Inter-arrival time distributions

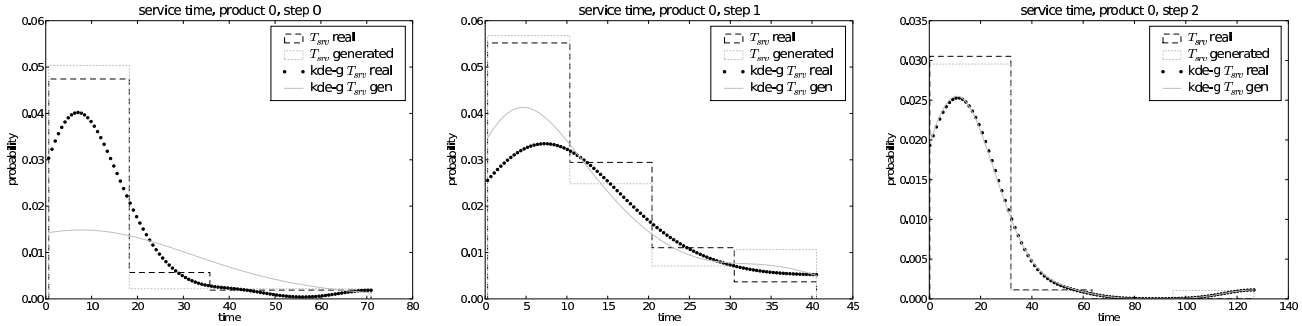

Fig. 10. Service time distributions

systems (up to 15 machines, 15 operations, 15 product types).

6. CONCLUSIONS

The results obtained here are encouraging. They validate the implementation approach as well as the tool that we have developed. One of the next stages of our work will take into account imprecise location measures, the variations in the sample period and their impacts in the generator. Finally, the generator in a prepared version will be confronted with a blind validation on a real system.

REFERENCES

Bahouth, A., Crites, S., Matloff, N., and Williamson, T. (2007). Revisiting the Issue of Performance Enhancement of Discrete Event Simulation Software. In *40th Annual Simulation Symposium (ANSS'07)*, volume 0, 114–122. IEEE Computer Society, Los Alamitos, CA, USA.

Cai, X. (2005). On the performance of the Python programming language for serial and parallel scientific computations. *Scientific Programming*, 13(1), 31–56.

Cassandras, C.G. and Lafortune, S. (1999). *Introduction to Discrete Events Systems.* Kluwer Academic Publishers, Dordrecht.

De Vin, L.J., Ng, A.H.C., and Oscarsson, J. (2004). Simulation-Based Decision Support for Maunufacturing System Life Cycle Management. *Journal of Advanced Manufacturing Systems*, 3, 115–128.

De Vin, L.J., Ng, A.H.C., Oscarsson, J., and Andler, S.F. (2006). Information Fusion for Simulation Based Decision Support in Manufacturing. *FAIM 2005 Special Issue of Robotics and Computer Integrated Manufacture*, 22, 429–436.

Downey, A., Elkner, J., and Meyers, C. (2002). *How to Think Like a Computer Scientist: Learning with Python.* Green Tea Press.

Karkkainen, M., Holmstrom, J., Framling, K., and Artto, K. (2003). Intelligent products - a step towards a more effective project delivery chain. *Computers in Industry*, 50, 141–151.

Mittal, S., Mak, E., and Nutaro, J.J. (2005). DEVS-Based Dynamic Model Reconfiguration and Simulation Control in the Enhanced DoDAF Design Process. *submitted to Journal of Defense Modeling and Simulation.*

Muller, K. (2004). Advanced systems simulation capabilities in SimPy. *Europython 2004.*

Muller, K. and Vignaux, T. (2003). SimPy: Simulating Systems in Python. *ONLamp.com Python Devcenter.*

Park, K.J. and Lee, Y.H. (2005). A On-line Simulation Approach to Search Efficient Values of Decision Variables in Stochastic Systems. *Int J Adv Manuf Technol.* doi:10.1007/s00170-003-1951-0.

Rossum, G. (1995). Python reference manual. CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands.

Thiesse, F. and Fleisch, E. (2008). On the value of location information to lot scheduling in complex manufacturing processes. *International Journal of Production Economics*, 112(2), 532–547.

Wilcoxon, F. and Co, A.C. (1997). Individual Comparisons by Ranking Methods. *Breakthroughs in Statistics.*