Optimizing Business Processes using Attributed Petri Nets

Bernd Eichenauer, IBE Simulation Engineering GmbH

Abstract: A method is described which allows the exact modeling and optimization of business processes under arbitrary constraints.

In the last years the optimization of business processes has gained increasing attention since it can be used to reduce costs at relatively low effort. Furthermore, it is the endmost opportunity at-hand to encounter the cost pressure that arises from the globalization of markets. In the following we introduce a method to create exact and executable models of discrete processes. The models can be optimized using mathematical methods. The introduced method has been proven in numerous applications (see, e.g., [1,2,3,4,5,6,7,8,9]).

1. Modeling Method

Today, systems for the planning of workflows and the automation of processes are frequently based on CASE methods introduced in the 1980s and 1990s. Their use is limited as they generally describe and verify the systems planned in an incomplete manner. Typically only the static parts of an application are characterized by a semi formal system specification using pseudo code and numerous graphics. Thus only the consistency of the specification can be verified but not its content. It are precisely the tasks where computers could be useful that are inadequately supported by most of the case tools available today, namely the verification of the dynamical aspects of a system specification.

A further drawback of this approach is the discrepancy between the (procedural) application view and the abstract model. Especially object-oriented modeling methods frequently lead to results that are hard to understand. The consequence is a lack of acceptance and of actual usage by the users for which the models were actually produced.

These and other drawbacks were the initial trigger for the search of a new modeling method that is both exact and user-friendly. The initial requirement was, that the user should be able to recognize his application in the model, which means that the model should be close to the real application. With other words: a model should be functionally as well as structurally a realistic map of the application.

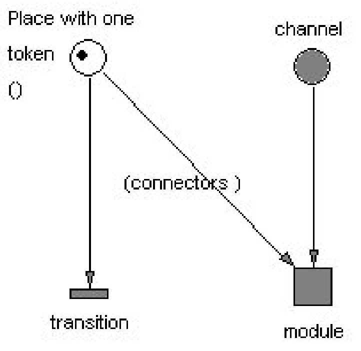
Structural and functional conformance between reality and model can be reached, if the visual structure can be identically mapped into the model. The postulation of identical behavior implies that the model has to be an executable simulation model. An executable (animated) model enables cost efficient experiments to verify the design of a system, or to optimize process parameters with respect to certain criteria like production costs.

The analysis of exact modeling of industrial applications led to the development of a semi-graphical modeling language, MSL[1], and its implementation in the simulation development environment PACE [10]. MSL provides three kinds of language elements, namely elements

- to describe the structure of an application including its process graph
- to visualize the objects that are moving along the process graph
- to process data objects and other process data.

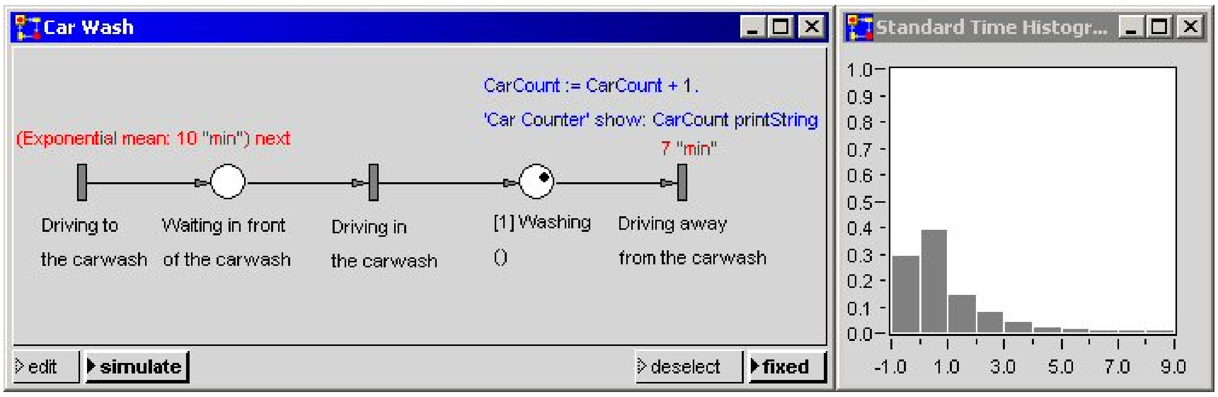
Elements from Petri nets can be used to visualize the process graph. In MSL process steps are described using generalized places, transitions, and connectors. The generalized elements are given attributes. These attributes determine the precise treatment of the objects at the nodes of the net.

Besides the well-known elements of the so-called S/T-nets, MSL provides two further net elements, channels and modules. These elements serve to describe the hierarchical structuring of nets. They can be used to map the structure of the real system to the model. Modules contain re-usable partial nets that can be inserted into an arbitrary net using places and channels as interfaces. Channels are similar to passive net elements just like places. They are used to combine several connectors between modules.



As in Petri nets, objects traveling along the process graph are visualized as tokens or containers. The attributes of the tokens characterize the entities of the real objects. In order to access the attributes of the objects they are assigned to connector variables attributed to the connectors. Processing (modification) of the objects and other process data is mainly done at the transitions containing instructions in a script language. At present MSL uses Smalltalk-80.

The following example of a car wash demonstrates the interaction of the different elements of the language. You can see the places and transitions, as known from S/T nets that determine the feed of cars within the model, regulate the time of the washing activity, or count and display the number of washed cars. The "Distribution Time Histogram" shows the probability to find a certain number of cars in the queue.



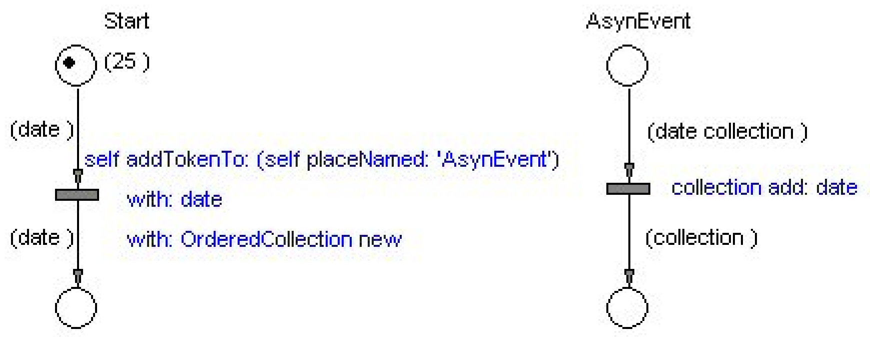
[1] Modeling and Simulation Language

2. Net Functions

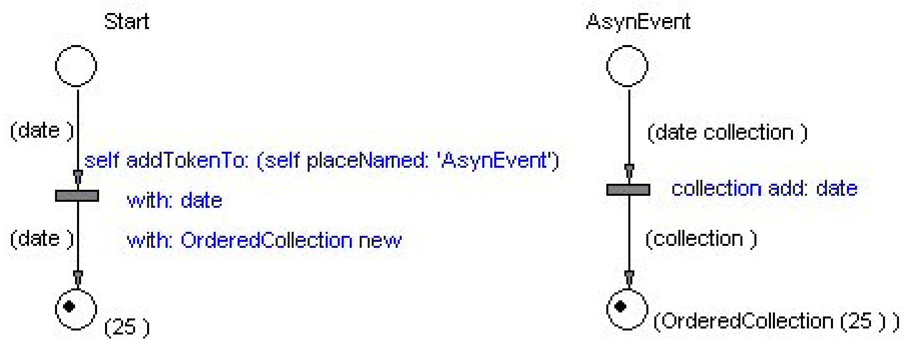
Giving attributes to the net elements provides many new ways of constructing nets. Hereby closed process graphs are of special meaning. They can be "called" with varying parameters from different transitions and return results to the calling environment. In this paper we will refer to these process graphs as "net functions". At first we will consider a design that can be used for such functions.

In real systems not only synchronous but also asynchronous events will occur. Conventional Petri nets do not provide the later. In MSL asynchronous events are generated by introducing a token, with or without attributes, into a place. In the MSL code we use the addTokenTo: message for this event.

Example:



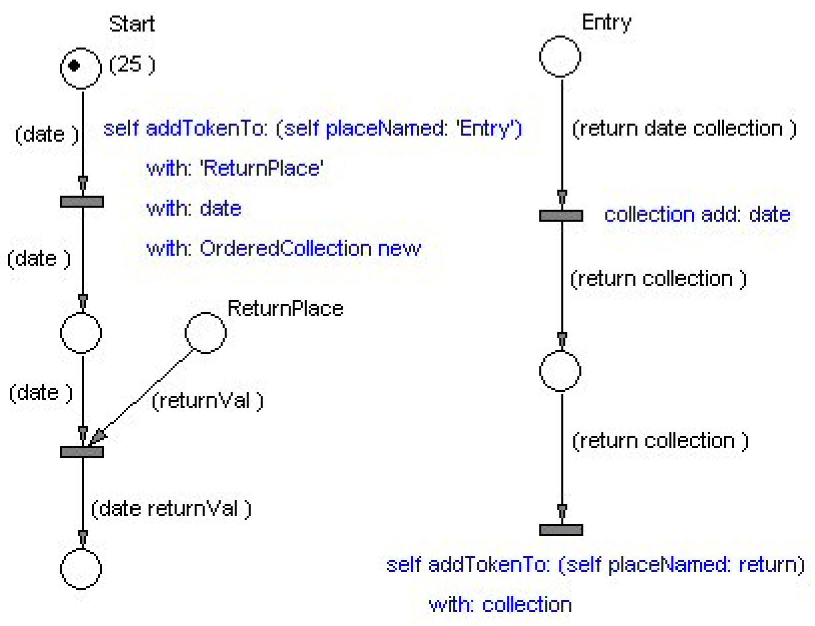
The figure shows two simple process graphs. Initially only the left graph contains one token. As soon as the simulation is started the process in the left graph adds a token to the place called "AsynEvent". This, in turn, starts a process in the right graph. The final state is shown in the following figure.



Start
(date)
self addTokenTo: (self placeNamed: 'AsynEvent')
with: date
(date)
with: OrderedCollection new
(25)
AsynEvent
(date collection)
collection add: date
(collection)
(OrderedCollection (25))

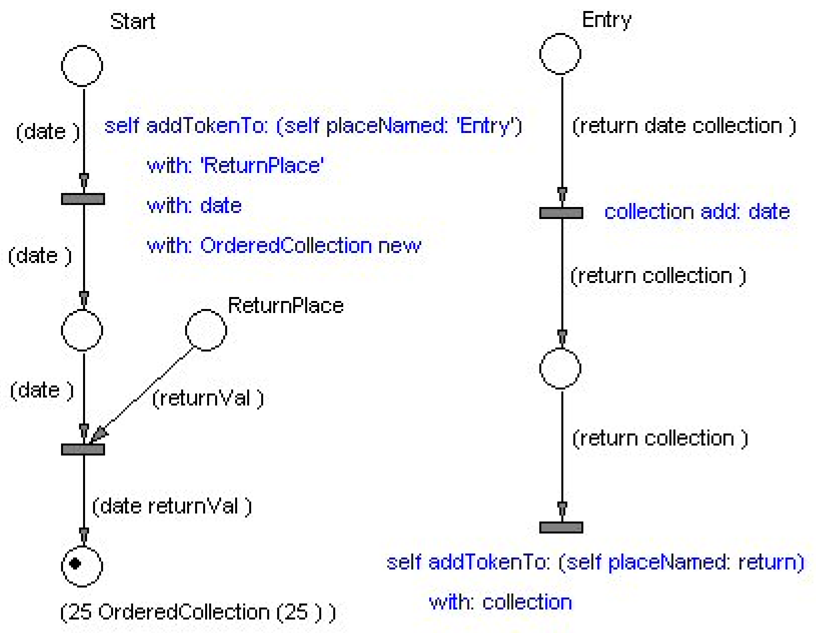
The described mechanism can be used for net functions as follows: The name of the place which accepts the return value and optional user data is passed to the net function. At the end of the net function the return value is handed over again using the addTokenTo: message.

Example:



The right hand side process graph shows the net function. The process displayed on the left starts the net function and hands over three parameters to the net function. The function result is returned to the place labeled “ReturnPlace”.

The resulting graph is shown in the following net:



3. Optimizer

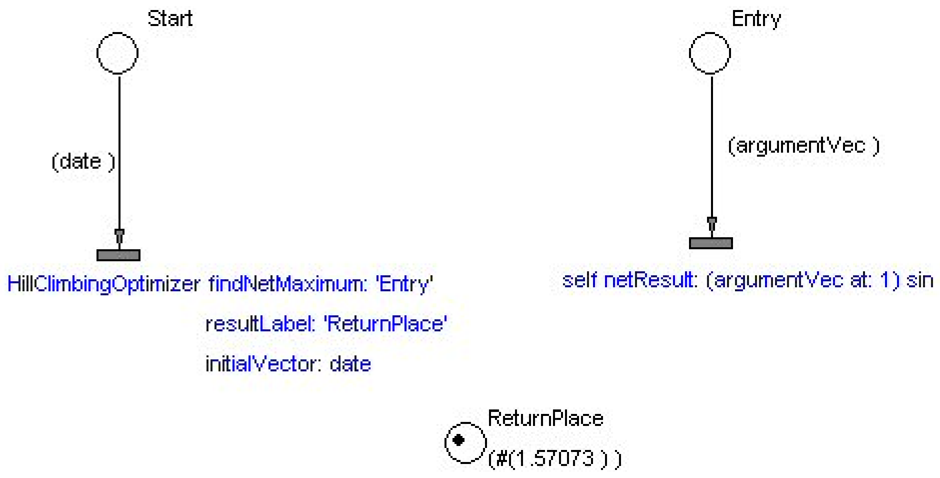
The availability of net functions leads to the question about extreme values of these functions. If we model a part of an entire business process as a net function, then the extreme of the net function would just represent the optimal parameters of this (sub) process.

The determination of the extremal values can be seen in analogy to the search for extremal values of ordinary mathematical functions. In general the analytical properties of the net function, however, are not known. This has to be considered when choosing the optimization procedure. In many cases they are generalized procedures that can only be evaluated for certain arguments. This case is frequently found when optimizing processes with resources that cannot be partially allocated.

Therefore, optimization methods where selected that require only the continuity of net functions If this is not the case the methods are extended by additional scaling mechanisms. Currently the hill climbing method, the simplex method, and a genetical algorithm are implemented for mathematical and net functions. The methods can be used individually or in sequence.

To be able to connect an optimizer the described mechanism for calling net functions must be slightly modified. The function call is no longer performed with the addTokenTo: message. When an optimizer is called the entry place of the net function and the place where the result has to be delivered is handed over. The netResult: message is used for returning the current function value to the calling optimizer.

Example:


Start
(date)
HillClimbingOptimizer findNetMaximum: 'Entry'
resultLabel: 'ReturnPlace'
initialVector: date
Entry
(argumentVec)
self netResult: (argumentVec at: 1) sin
ReturnPlace
(#(1.57073))

Starting at a value of 0.2 the maximum of the sinus function is calculated using the hill climbing method.

4. Case Study

4.1 Nature of the Task

In the following the described optimization method is demonstrated using a more extensive example. Consider a production facility, which is separated into two parts, preparation and production. Five different products with production parameters given in the following table are produced.

In the columns you can find the parameters that have to be evaluated during the simulation:

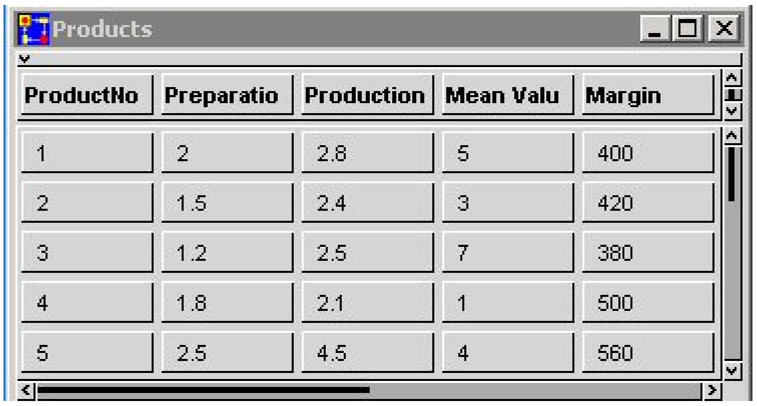
Column 1: Product number

Column 2: Time in hours required for the preparation.

Column 3: Time in hours required for the production.

Column 4: Average time span between two sales.

Column 5: Raw revenue = sales price – material costs without costs of labor



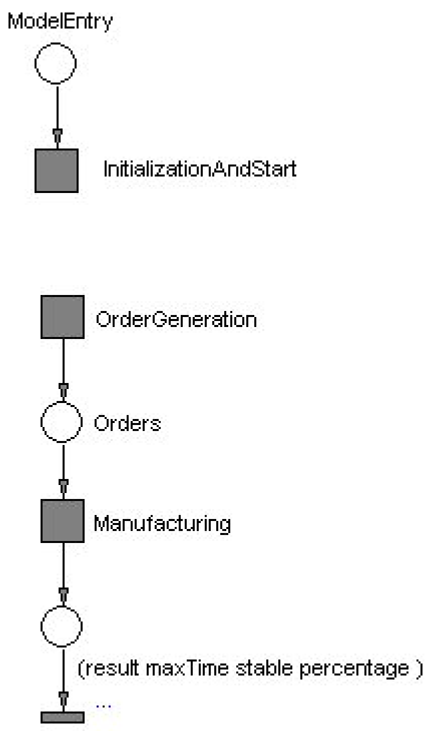
Products

ProductNo	Preparatio	Production	Mean Valu	Margin
1	2	2.8	5	400
2	1.5	2.4	3	420
3	1.2	2.5	7	380
4	1.8	2.1	1	500
5	2.5	4.5	4	560

In addition, the labor costs for preparation are estimated at 30 euros/hour. One hour of labor in the production department is estimated at 40 euros/hour.

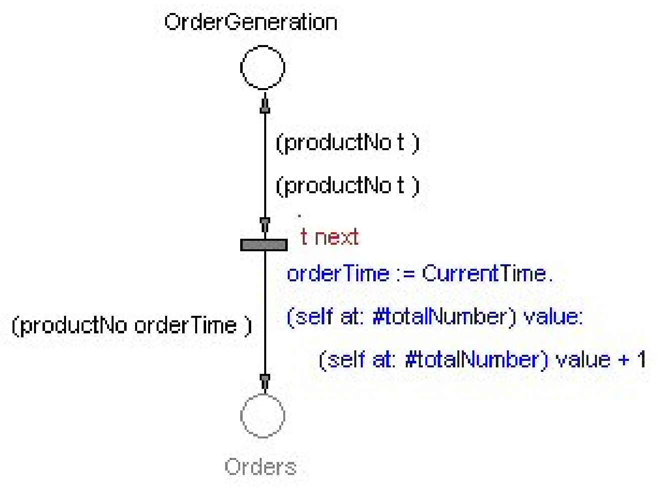
4.2 Modeling

The figure on the right side shows the top hierarchical level of the net function. More detailed levels can be shown when opening the modules, which contain increasingly detailed implementation details. However, these are normally not shown on the application surfaces of simulators.



The module "ProductGeneration" consists of two parts, the module “InitializationAndStart” and the module where the production is modeled. The latter is a combination of the departments “OrderGeneration” and “Manufactoring”. The function is called by inserting a token with two parameters into the place “ModelEntry”. The parameters contain the number of personnel in the preparation and in the production. The function result is the profit per hour, which is calculated by the simulator over a sufficiently large time interval (e.g., 1000 hours).

The initialization is not discussed in detail here. It sets the number of personnel in the different departments and ensures that at the start of the production identical initial conditions exist in the preparation and production. More interesting is the start of the production. It is triggered by inserting 5 tokens into the place “OrderGeneration” of the module with the same name. Accordingly, deleting the token in the place “OrderGeneration” terminates the production. In this way it is easy to model a switch using the message addTokenTo:, and the message for deleting all tokens.


OrderGeneration
(productNo t)
(productNo t)
t next
orderTime := CurrentTime.
(self at: #totalNumber) value:
(self at: #totalNumber) value + 1
(productNo orderTime)
Orders

Each token in the place “OrderGeneration” has two attributes, the product number and an exponential distribution, whose mean value is determined by the average time between two orders for the product as shown in the previous table. The connector assigns the distributions to the variable “t”. The next value of the distribution “t next” is interpreted as a waiting time until the next transition. As soon as the waiting time is over a token with two attributes, the product number “productNo”, and starting time “orderTime” is generated and transferred to the place “Orders”.

The place “Orders” is used as an interface between the modules “OrderGeneration” and “Manufacturing”. It should be noted that it is drawn here with a shade of grey as compared to the last but one figure (Module "ProductGeneration"). This indicates that the place is defined on a higher level of hierarchy.

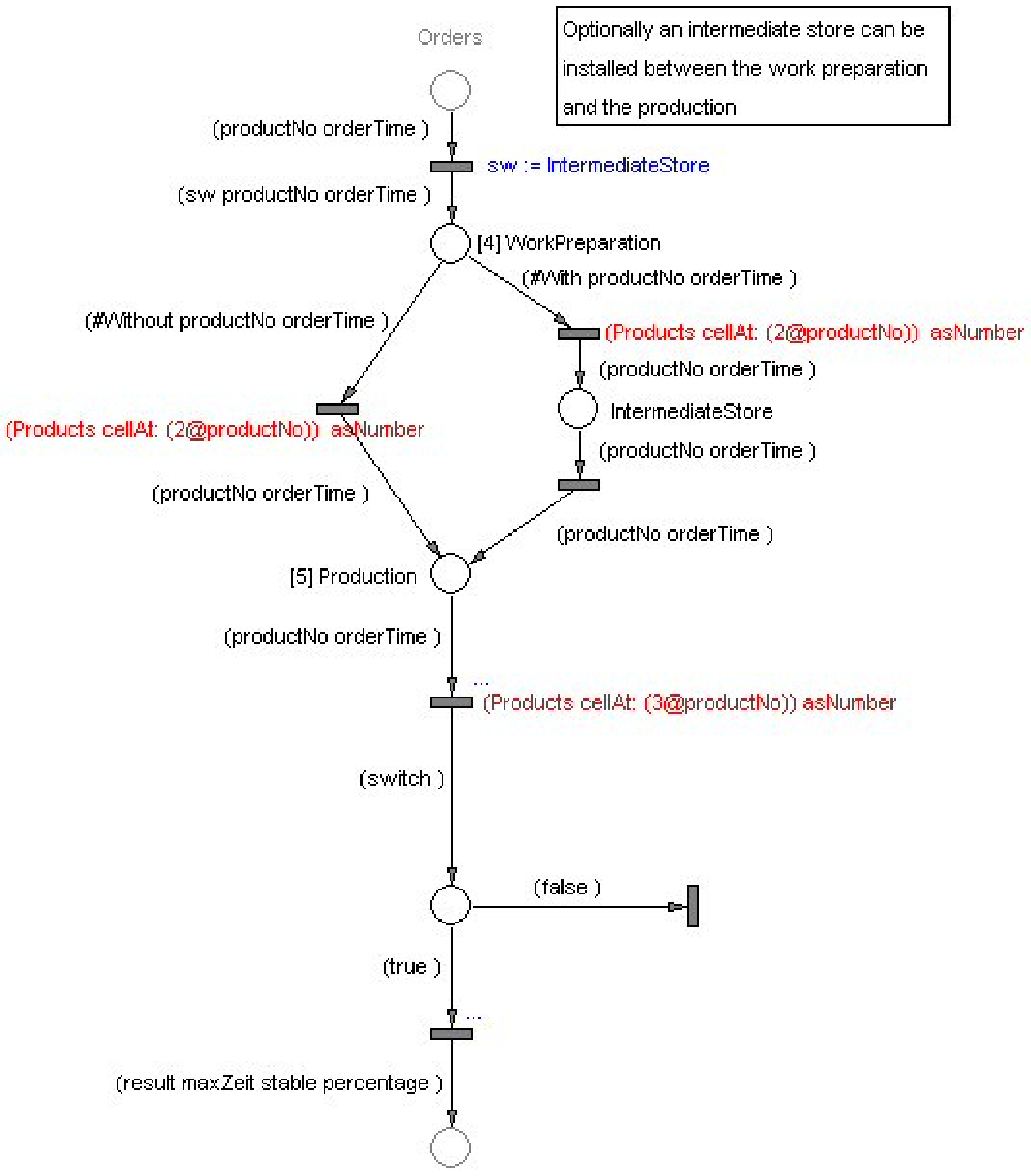
The figure on the next side shows the simplified module “Manufacturing”.

The queues in front of the two departments are organized in the places “WorkPreparation” and “Production”. The values in square brackets are handed over by a call of the net function and indicate the capacity of the places which represent the actual number of workers. The waiting resp. work time inscribed in the following transitions is defined in column 2 and 3 of the above table “Products”. The module accumulates the revenue during a period of 1000 hours. After this time the result is calculated from the difference of the revenue and the cost for personnel.

4.3 Experiments with the Model

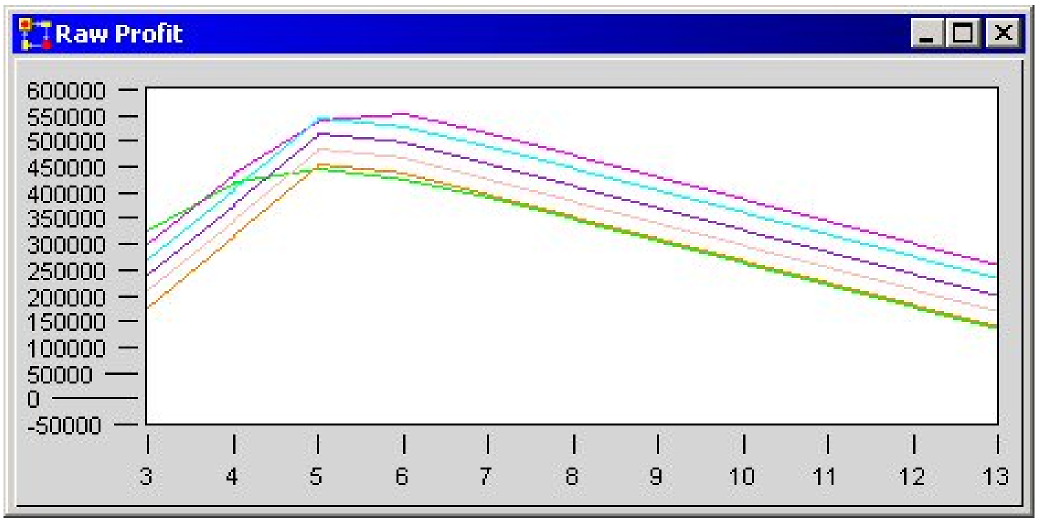
The description in the previous chapter shows the most important features of the business model given in section 4.1. To keep it short, the model was somewhat simplified. The extended net function can be used both as a standard net function and in combination with the optimizer. In addition, it allows to define additional constraints, e.g., to purport a maximum processing time for the execution of an order. The experiments described in the following use the extended model.

The most important question for the layout of a modeled business process is how to pick the resources in order to gain the largest possible profit. From the table “Products” the personnel employed in both departments would sum up to 3.5 workers for preparation and 4.95 for the production. Considering only these mean values one would employ 4 respectively 5 skilled workers in these departments.

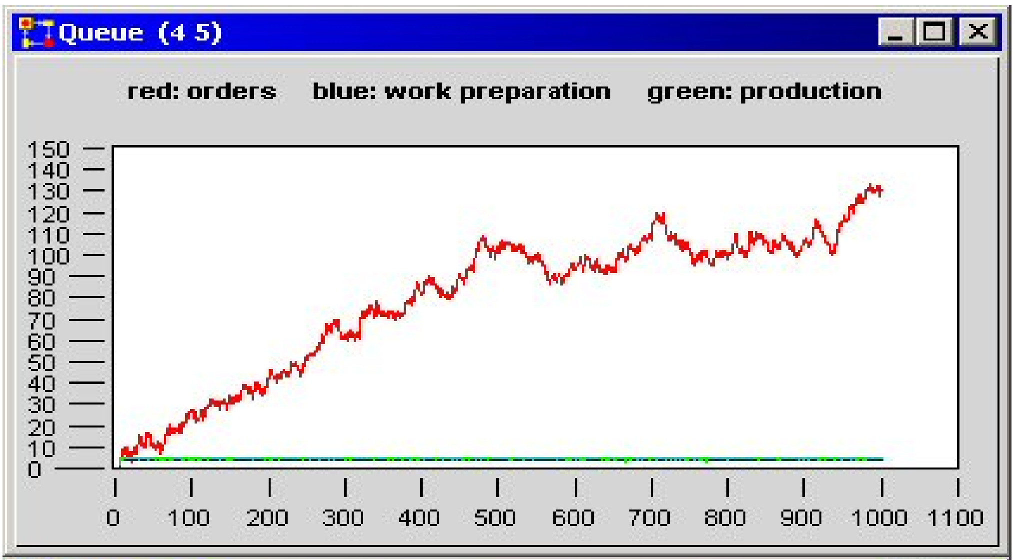

Orders
Optionally an intermediate store can be installed between the work preparation and the production
(productNo orderTime)
sw := IntermediateStore
(sw productNo orderTime)
[4] WorkPreparation
(#With productNo orderTime)
(#Without productNo orderTime)
(Products cellAt: (2@productNo)) asNumber
(productNo orderTime)
IntermediateStore
(Products cellAt: (2@productNo)) asNumber
(productNo orderTime)
(productNo orderTime)
(productNo orderTime)
[5] Production
(productNo orderTime)
...
(Products cellAt: (3@productNo)) asNumber
(switch)
(false)
(true)
...
(result maxZeit stable percentage)

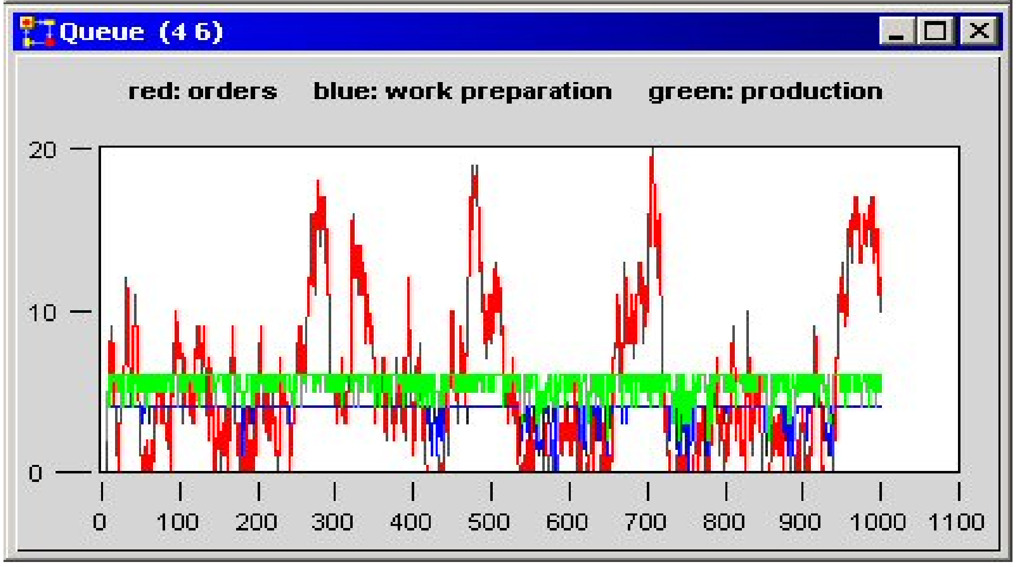
In the underlying case of two departments the optimal number of workers can be determined graphically. The profit accumulated over 1000 hours by the net function is calculated and visualized in two nested loops. The result is shown in the following figure.

The x-coordinate shows the number of personnel in the production. The number of workers in the preparation department corresponds to the color of the respective graph (3:green, 4:magenta, 5:cyan, 6:purple, 7:pink, and 8:orange) or by counting the curves on the ordinate top down. It can be seen that the optimum is not found at the mean values (4,5) but at (4,6).



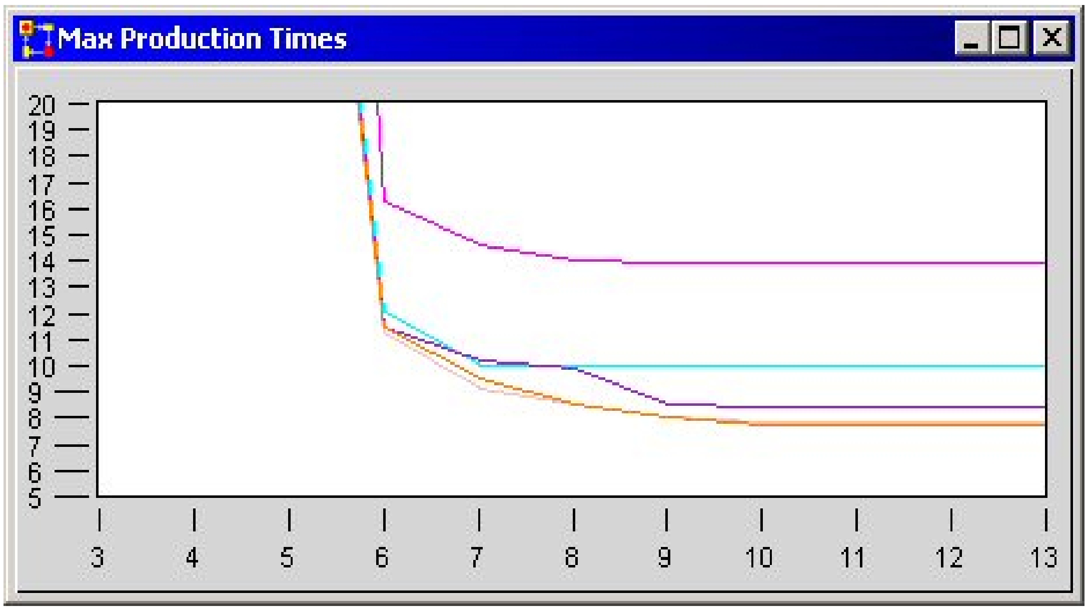
It is also interesting to take a look at the behavior of the queues in the departments. It can be seen by the behavior of the order queues as functions of the time and is displayed for the values (4 5), and (4 6) in the following two figures.





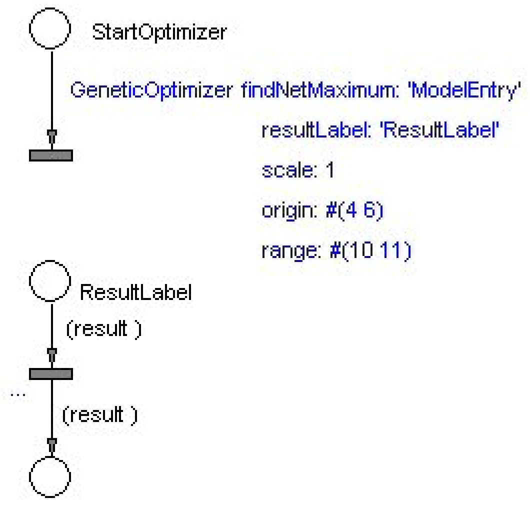
The figures show that with only 5 workers the order queue is not stable, i.e., the production time (run time) of an order is dependent on the time it was started. However, the production is rendered stable if the number of production workers is increased from 5 to 6.

The numbers given here will change as soon as the production time for an order is limited. Constraints like these arise when there is no store available for the resulting products. In the next figure the maximum production times of orders are displayed for a range of 4 to 8 workers in the preparation department. The graph corresponding to a certain number of workers can be extracted from the color or found by counting the graphs on the right side from top to bottom. For the maximum production times of 8 resp. 9 hours we see that (7 10) resp. (7 8) workers are required.



4.4 Use of an Optimizer

The figure shows the interface between the module "ProductGeneration" described in Section 4.2 and the optimizer. Here a genetic optimizer is used for searching the optimum in a range from (4 6) to (10 11). Only integer numbers are considered (scale: 1).

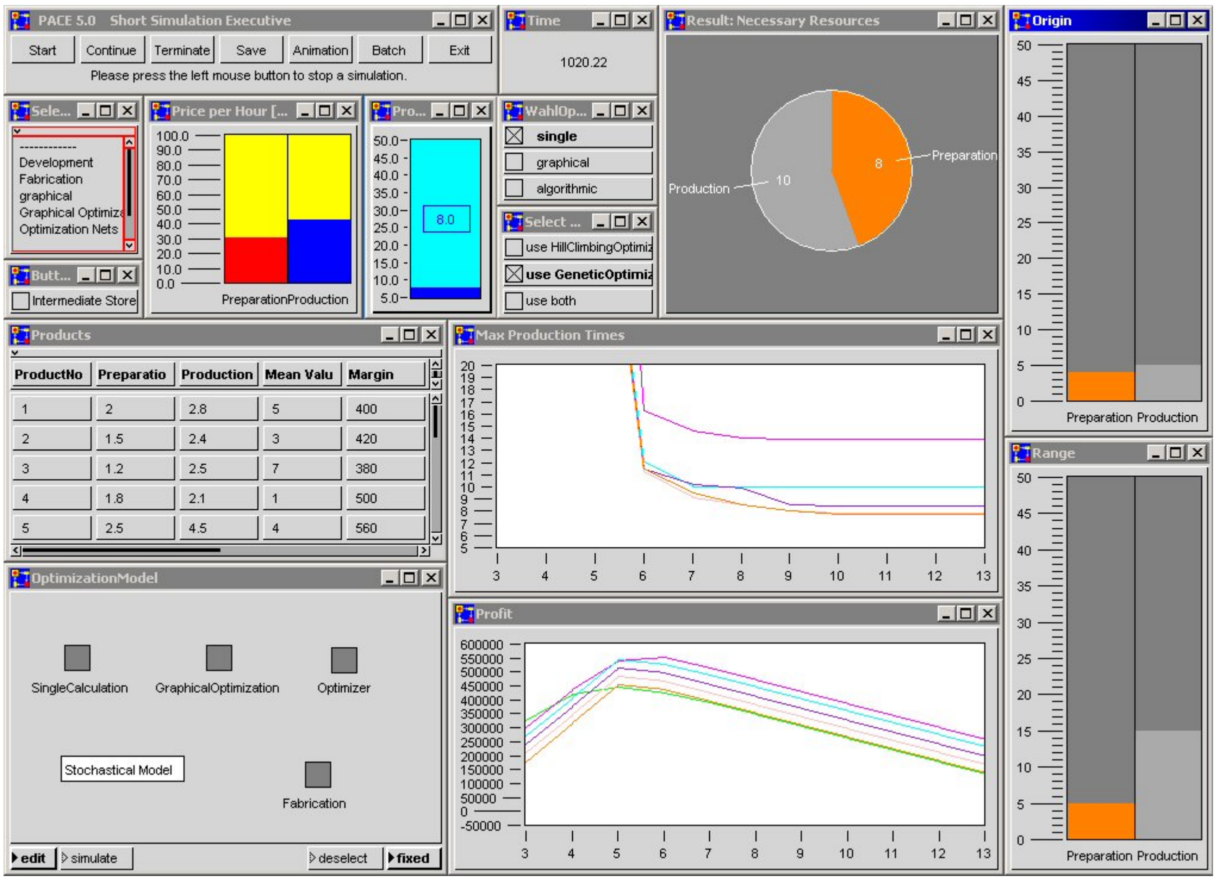


In order to limit the production time for the tasks the module "Production" was modified as follows.

The maximum production time of an order found in the simulation is divided by two if the requested maximum production time is exceeded during the simulation within 1000 hours. A maximum production time of 8 and 9 hours yields the same result as previously obtained from the graphical examination.

4.5 User interface

To ease the use of models, PACE includes numerous graphical and textual input and output elements. They can be easily put together to a user interface that can be used also by persons that were not involved in developing the model after a short briefing. In the case study at hand the user interface shown in the following figure was used.



Please press the left mouse button to stop a simulation.

1020.22

Development
Fabrication
graphical
Graphical Optimiz
Optimization Nets

[ ] Intermediate Store

[✓] single
[ ] graphical
[ ] algorithmic

[ ] use HillClimbingOptimiz
[✓] use GeneticOptimiz
[ ] use both

8.0

ProductNo	Preparatio	Production	Mean Valu	Margin
1	2	2.8	5	400
2	1.5	2.4	3	420
3	1.2	2.5	7	380
4	1.8	2.1	1	500
5	2.5	4.5	4	560

The model can be operated using the “Short Simulation Executive” shown in the upper left corner. Initial values and model parameters described earlier are set by bar gauges or read from the table. Depending on the selected mode (single, graphical, algorithmic) the result is presented as a pie chart or in form of curves.

5. Conclusion

The introduced modeling and optimization method is not limited to business processes but can be used for arbitrary discrete processes. The method has already been applied in traffic control, construction, and logistics. The method has been incorporated into the simulator development environment PACE 5.0 for a systematic approach to system optimization.

Literature:

[1] C. Böhnlein: Modellierung des Bullwhip-Effekts mit Hilfe höherer Petri-Netze, wisu, 31 (2002) 8-9, S. 1124-1127

[2] U. Dietel, F. Bennemann: Reihenfolgebildung von Gießaufträgen bei einem sächsischen Metallhersteller, in: M. Rabe, B. Hellingrath: Handlungsanleitung Simulation in Produktion und Logistik – Ein Leitfaden mit Beispielen für kleinere und mittlere Unternehmen, SCS International, 2001, ISBN 1-56555-226-1

[3] U. Dietel, H.-J. Hanisch: Simulation des Bereichs Wärmebehandlung in der Porzellanherstellung, in: M. Rabe, B. Hellingrath: Handlungsanleitung Simulation in Produktion und Logistik – Ein Leitfaden mit Beispielen für kleinere und mittlere Unternehmen, SCS International, 2001, ISBN 1-56555-226-1

[4] S.M.O. Fabricius, E. Badreddin: Stochastic Petri Net Modeling for Availability and Maintainability Analysis, Proceedings of 14th International Congress and Exhibition on Condition Monitoring and Diagnostic Engineering Management (COMADEM), 2001, Manchester, UK

[5] V. Franz: Production simulation techniques in practice, BFT 5, 2001, S. 20 – 23

[6] B. Eichenauer, K. Scherer: Modellierung und Simulation von intelligenten Gebäudesystemen mit attributierten Petri-Netzen, ikm 16. Internationales Kolloquium über Anwendungen der Informatik und Mathematik in Architektur und Bauwesen, Weimar, Juni 2003

[7] M. Enkelmann: Simulation in block-production operations, BFT 5, 2001, S. 24 – 28

[8] B. Eichenauer: Simulation, the key to more efficiency in commercial and production processes, BFT 6, 2001, S. 40 - 44

[9] G. Feistl: Findet das Nadelöhr – Simulation von Verpackungsprozessen, neue verpackung 4, 1998, S. 32 - 34

[10] PACE 5.0 User Manual, IBE Simulation Engineering GmbH, 2002, www.ibepace.com