Optimization of Process Chain Models with Response Surface Methodology and the ProC/B Toolset

Peter Buchholz, Dennis Müller, Axel Thümmler

University of Dortmund, Informatik IV, 44227 Dortmund
http://ls4-www.cs.uni-dortmund.de/MuS

Abstract

Large logistics networks often require sophisticated decisions to be made to meet the required service qualities. Often these decisions are made according to a model based analysis and optimization of the network. For this purpose simulation models and appropriate optimization techniques have to be combined. This combination is still a challenge, in particular if the approach should run in a more or less automated way.

In this paper we present the combination of a process chain based simulator and the response surface method for optimization. Particular emphasis is placed onto a realization of the response surface method which runs completely automatically after initialization. The quality of the proposed optimization approach is shown by means of two example models.

1 Introduction

The design and operation of logistics networks requires a large number of decisions to be made to find a realization of the network which meets the required service qualities and which can be realized with low costs. Usually several design parameters like the location and capacity of depots, the number and type of vehicles and other resources have to be set to the right values to meet the design goals. Due to the complexity of the problem one cannot expect to find an optimal or at least a good solution in an ad hoc manner. What is required is a systematic approach which can be applied without too much effort. Such an approach has to be model driven since experiments using the real system are impossible or too costly.

The inherent complexity of logistics networks requires an adequate modeling formalism to describe them and adequate methods to analyze them. For the analysis part discrete event simulation often has to be the method of choice since it is the only approach that can be applied for the analysis of large and complex models as they result from real scenarios. For model specification different approaches exist, often process chains are used since they allow a very natural mapping of real world processes onto the corresponding modeling elements [7], [12]. However, usually process chains are only descriptive models and cannot be used as specification of a simulation model. Consequently, process chains have to be enhanced with additional information to allow an automatic mapping onto a simulation program which can be used to determine quantitative results like processing times, filling of depots, down time of vehicles or costs of the whole design. One modeling approach that is based on process chains and can be used to specify simulation models is the ProC/B approach with the corresponding toolset [1] which has been developed in the collaborative research center 559 [5] at the university of Dortmund. The ProC/B Toolset allows the graphical and hierarchical specification of process chain models, the subsequent mapping onto different analyzable models including simulation programs and the representation of the results at the level of the process chain description.

With the ProC/B Toolset models of logistics networks can be specified and analyzed, but the finding of optimal or good designs is not supported in the original toolset. From an abstract mathematical point of view the simulator represents a function $\phi(w_1,\ldots,w_k)$ for some input parameter vector $\mathbf{w} = (w_1,\ldots,w_k)$. Due to the stochastic nature of the simulation model $\phi(\mathbf{w})$ can only be observed with some statistical fluctuation and confidence intervals can be computed [8]. The optimization goal is to find $\min_{\mathbf{w}\in W} E[\phi(\mathbf{w})]$ where $E[\phi(\mathbf{w})]$, also denoted as $f(\mathbf{w})$, is the expectation of $\phi(\mathbf{w})$ and W is a feasible range for the parameters. Since ϕ is only implicitly represented as a simulation model or in other words as a *black box*, only those optimization methods can be applied which do not exploit the structure of function f. Optimization of simulation model is an important topic, several methods have been developed and some optimization packages are available [6], but most available optimization packages are not fully integrated with the simulation model, are not robust or do not scale well for larger problems. In so far the development and realization of integrated and automated optimization methods is a challenging research topic.

Among the large number of available optimization methods for simulation models the response surface method (RSM) [10], [11] became very popular and is often used. RSM originally has been developed for optimization based on real experiments, but can be easily extended to be used with simulation. However, although RSM is known for a long time and a well established theory has been developed, the algorithm is usually described in a way that several steps have to be done manually such that the approach cannot be integrated in an optimization package where the whole experimentation and optimization approach is done automatically. Only very few implementations of RSM exist which can be used for automatic optimization [11] and it is not trivial to realize robust implementations of RSM that work for a wide range of models.

In this paper we describe the extension of the ProC/B Toolset by an optimization package which applies RSM for the optimization of complex process chains. Of particular importance is the concrete realization of RSM such that it can run with very limited information or support by the user. In this way the toolset supports the whole design process of large logistics networks from the specification of the model to the finding of a good parameterization.

The paper is organized as follows. In the next section a brief overview of the ProC/B approach and the corresponding toolset is given. Afterwards RSM as an approach for the optimization of ProC/B models is introduced. Particular emphasis is given to the introduction of a realization of RSM that runs without interaction with the user. In Section 4, the optimization of two example models is represented, one small example, where the complete response function can be computed and one large and realistic model of a cargo transfer station. The paper ends with the conclusions.

2 Modeling Process Chains with the ProC/B Toolset

Process Chains (PCs) are a convenient paradigm to model complex scenarios of logistics networks. As a mixture of graphical and textual description they are often easy to understand, but on the other hand they are usually not completely formalized and therefore do not include a full description of the system dynamics which is necessary for a model based quantitative analysis. Available formal modeling approaches are directly analyzable, but are often more abstract and harder to understand for a modeler, especially if he or she is coming from an application area. Often a trade off between the degree of freedom the modeler has and the strictness of the modeling paradigm exists. However, even if formal models have to be more

more strict in their syntax and semantics, they can be designed in a way that they adopt common modeling elements from an application area and become in this way understandable and acceptable by people from that application area.

The ProC/B modeling paradigm is a formal approach for the specification of process oriented models of logistics networks which has been defined with two goals in mind:

1. Even complex models should be easy to understand and the modeling elements should be similar to known approaches in the area.
2. Models have to be analyzable by computers according to various analysis goals including quantitative evaluation of technical and economical measures.

The model class of ProC/B will be described in more detail in Section 2.1. The ProC/B Toolset is based on the ProC/B modeling paradigm. It includes a graphical editor to specify models and several converters to translate ProC/B models to various analysis tools like the HIT simulator [4]. The toolset is introduced in Section 2.2.

2.1 Description of the ProC/B Paradigm

The ProC/B paradigm allows a modular and hierarchical description of process oriented models. Processes are described using process chains (PCs) which are a graphical way to visualize and specify the behavior. Each PC (see e.g. Fig. 5) describes one or several behavior patterns realized by connected activities. Activities are represented graphically with some additional textual annotation. Activities are connected by so called connectors. Different connectors exist to realize common behaviors like conditional branches, fork-joins, or loops. Each behavior pattern starts with a source denoted as a circle with an included dot and ends in a sink denoted as a circle with an included “×”. In the process description of the example in Fig. 5, sources generate activities according to some (stochastic) description of the arrival process and are denoted as unconditional sources. Alternatively one may define conditional sources that are driven by some external process (see e.g., the introduction of hierarchies below).

Activities often have some duration and require some resources. To describe a pure delay or timeless data manipulation, activities can be adequately enhanced. For the specification of resource usage activities are enhanced by resource and service names (see Fig. 5). Resources are specified as *Functional Units (FUs)* which are named elements that are able to perform some named services. Each description of a FU contains one or several named services which are graphically represented by source and sink symbols at the border of the graphical representation (see the lower parts of Fig. 5). FUs describe the consumption of the basic resources time and space. One can distinguish between basic and constructed FUs. The ProC/B paradigm defines two basic FUs *Servers* and *Counters* for time and space consumption, respectively. Servers capture the behavior of traditional queues including different scheduling strategies and load dependent speeds. Counters realize the consumption of space and are realized by vectors of integer variables with individual upper and lower bounds. A request to a counter is immediately granted, if the result is within the predefined bound, otherwise the calling activity has to wait until the request becomes possible. Apart from simple ones, FUs may be constructed by defining their behavior using a process chain description including conditional sources and sinks. Each source/sink pair describes a service which can be used by some calling activity. From some upper level basic and constructed FUs look similar from the outside. By construction of FUs from PCs two hierarchies are defined, a behavioral hierarchy of process patterns and a structural hierarchy by the definition of FUs using other FUs. For a flexible use of the approach it is possible to define general (non tree-like) hierarchies by using FUs from another PC in one PC. The corresponding services are denoted as external. Of course, recursive usage of this construct is not allowed since it results in components defined by themselves.

Fig. 5 shows a simple example of a ProC/B model containing a single behavior pattern. Processes are generated with negatively exponentially distributed interarrival times and perform several activities. The first activity, named `Queue1`, uses a service `alter_or_skip` of a FU named `Unit`. Depending on the outcome of this activity the process performs additional activities or terminates immediately in the `else`-branch. Service `alter_or_skip` has three parameters and a return value which is stored in the local variable `success`. Depending on the value of `success`, the `if`- or the `else`-branch of the subsequent connector are chosen. Observe that the `Unit1` provides further services which are not all used by the PC of the model, but they might be used as external services by some other process description in the model. Of course, this model is very simple and contains only a few features, but it shows the general principle of modeling with PCs. Further details about the ProC/B approach can be found in the literature [1], [2].

2.2 The ProC/B Toolset

The ProC/B Toolset is a collection of tools which include an editor for PCs and several tools for quantitative analysis and visualization of the result measures. In this paper only the ProC/B editor and the simulation-based model analysis via the tool HIT will be briefly described.

The ProC/B Editor

With the ProC/B editor PCs can be modeled following the ProC/B modeling paradigm. The main window of the editor presents the current model in a hierarchical view, which allows the modeler to get an overview of his or her model and to access model parts directly.

The ProC/B editor supports two modes, the *modeling mode* and the *experiment mode*. In the modeling mode PCs are edited. A selected model part will be opened in a new window similar to the one shown in Fig. 5. The experiment mode allows the definition of measures and values of global variables. Measures can be defined for FUs and include the predefined standard measures throughput, turnaround time, population and utilization. By default these measures are evaluated globally, but it is also possible to obtain results according to a specific calling instance. Apart from the mentioned technically oriented measures it is also possible to define and evaluate economic measures by assigning costs to specific activities.

For optimization of models it is necessary to evaluate the model for several parameter settings to obtain an optimal or good design. Series of experiments are supported by the toolbox using global variables for the values that are modified during optimization and setting the values either in a predefined way or, as usually necessary for optimization, by some optimization algorithm that determines new parameter values from the results of the current experiments.

Simulation of ProC/B Models with HIT

As already mentioned ProC/B models can be transformed into models for several other tools, the resulting models can be analyzed with these tools and the results can be mapped back to the ProC/B model. Our optimization approach uses simulation as analysis method which can be done with HIT [4]. HIT is a powerful software tool originally developed for the analysis of computer and communication systems. It contains a simulator including support for statistical analysis of simulation results. Simulation runs can be controlled by different parameters including stopping criterions depending on the estimated accuracy of result measures.

For the coupling of optimization and simulation it is most convenient to access ProC/B models at the level of the HIT model. Parameters of the HIT model are modified by the optimization module and simulation control values are set. HIT produces results based on these settings and the results are interpreted by the optimization module. Only the final parameter set and the corresponding results (i.e., the computed optimal configuration) is translated to ProC/B.

3 Model Optimization with Response Surface Methodology

This section describes the framework for fully automated application of the *Response Surface Methodology* (RSM) for optimization of stochastic simulation models. In general, RSM is a collection of statistical and mathematical techniques, which are applied for the optimization of stochastic functions [10]. RSM relies on low-order linear regression metamodels. Local marginal effects of the simulation model are estimated to find a direction of improvement. Fig. 1 shows a possible course of the general RSM procedure in a two-dimensional search space. In the local-exploration phase (see phase I in Fig. 1), RSM uses a sequence of first-order regression metamodels, combined with steepest ascent search. In this phase four points in a square are simulated and a first-order regression model is approximated to characterize the response surface around the current center point. In the final optimization phase, RSM uses a second-order regression metamodel (see phase II in Fig. 1) to estimate the optimum from the resulting fit.

As already mentioned, a simulation model can be represented by a stochastic function ϕ, which maps a set of input parameters $w_1, \ldots, w_k$ onto a sample of an output performance measure which is a random variable. The goal of optimization is the optimization of the expectation of this random variable which is expressed by

$$f(w_1,\ldots,w_k) = E\left[\phi(w_1,\ldots,w_k)\right] = y \tag{1}$$

where f is also called the *response surface function*. Note that a simulator produces a sample of the output random variable according to a given seed value of a particular random number generator implemented in the simulator. In general, the expected value of this random variable is determined by replicating the simulation run for different seed values.

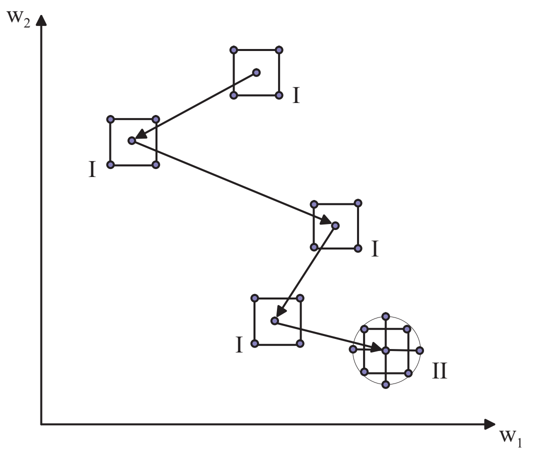

Fig. 1. Illustration of the Response Surface Methodology in two dimensions

The general optimization problem discussed in this section is characterized by finding a setting of the input parameters that maximizes/minimizes the response surface function. In RSM, the input parameters of the simulation model are usually called *factors*, whereas the stochastic output is called the *response* of the simulation model. Note, that in general input factors can be quantitative (i.e., continuous) and/or qualitative (i.e., discrete) variables. Nevertheless, in the following we assume that the input factors are continuous variables only, but it should be mentioned that our approach can be applied in a similar way for a mixture of continuous and discrete variables (see [10], pp. 456-478, for a discussion on qualitative variables). The main steps of the proposed RSM optimization algorithm are the following:

(i) Approximate the response surface function in a local region by a low-order linear regression metamodel
(ii) Test the metamodel for adequate approximation
(iii) Use the metamodel to predict the factor values of improved response

Note that steps (i) to (iii) are repeated until a certain stopping criterion is reached. In fact, after step (iii) is finished and an improved response has been determined, a new regression metamodel is approximated in the local region around the improved response. Steps (i) to (iii) are described in more detail in the next sections.

3.1 Approximating the Response Surface Function by Regression Models

In this section the approximation of the response surface function in a local region by a low-order linear regression metamodel is considered. In particular, we consider first-order and second-order regression models and develop the equations required for implementing the RSM algorithm. A comprehensive introduction in regression metamodels and a detailed development of the required theory can be found for example in the textbook [10].

In general, the units of input parameters $w_1, \ldots, w_k$ of the simulation model differ from each other. Even if some of the parameters have the same units, not all of these parameters will be considered over the same range. For example, one parameter may be the mean arrival rate of customers to a queue, ranging from 0.1 to 10, and a second parameter may be a failure probability of the server, ranging from 10^{-5} to 10^{-2}. Since input parameters have different units and/or different ranges in the experimental setting, regression analysis should not be performed on the raw (dimensional) parameters themselves. Instead, the input parameters must be *normalized* before performing the regression analysis. Thus, they are called *normalized variables* or *coded variables*. Each of the coded variables is forced to range from −1 to 1. Let l_i and u_i the lower limit and upper limit of input parameter w_i, $i = 1,\ldots,k$, respectively. We denote the coded variable that corresponds to the natural variable w_i by x_i. The transformation from w_i to x_i is performed by

$$x_i = \frac{w_i - m_i}{b_i} \tag{2}$$

with the center and half-width of the considered range $m_i = (u_i + l_i)/2$ and $b_i = (u_i - l_i)/2$, respectively.

The first-order regression metamodel in the coded variables is given by

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i \cdot x_i + \varepsilon \tag{3}$$

with k+1 *regression coefficients* $\beta_0, \ldots, \beta_k$ and ε an additive error having normal distribution with mean zero and variance σ^2. Estimators of the regression coefficients are determined using ordinary least-squares (OLS) estimation. Suppose that $n > k$ observations of the response variable, denoted by $y_1, \ldots, y_n$, are available. Each observed response y_i is the result of a simulation experiment with input factor values $x_{i1}, \ldots, x_{ik}$. Note that the input factors must be transferred to the natural variables to carry out the simulation experiment. Choosing an appropriate set of input factor values $x_{i1}, \ldots, x_{ik}$, $i=1,..,n$, is called an *experimental design* [9]. With OLS estimation the regression coefficients are determined such that the sum of squares of the errors ε_i for each parameter/response pair $x_{i1}, \ldots, x_{ik}, y_i$ in Eq. (3) is minimized. Writing the OLS system in matrix notations yields

$$\mathbf{y} = \mathbf{X} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{4}$$

with

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \ \mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{pmatrix}, \ \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix}, \ \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \tag{5}$$

The least-squares estimates, $\hat{\boldsymbol{\beta}}$, of the regression coefficients $\boldsymbol{\beta}$ are computed by

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y} \tag{6}$$

Experimental designs, where the regression matrix $\mathbf{X}$ satisfies $\mathbf{X}^T \cdot \mathbf{X} = n \cdot \mathbf{I}$, are called orthogonal. Orthogonal designs simplify the computations, since they lead to uncorrelated regression coefficients and to minimal variance of the predicted response in the region of interest. In the proposed RSM algorithm, we use an orthogonal design with two levels for each factor for the approximation of first-order regression models, i.e., a *2^k factorial design* or a *2^{k-p} fractional factorial design* [9]. To approximate a second-order model a 2^k factorial design is augmented with n_c center points and 2k axial points yielding a *central composite design* (see Fig. 2).

Linear regression in the widest sense is based on regression metamodels which are linear in the regression coefficients but not necessarily linear in the coded variables. In linear regression the computation of the regression coefficients is always based on the solution of Eq. (6) which results from minimizing the error sum of squares.

Recall from Section 3.1, that the final optimization phase of the RSM algorithm is based on second-order regression metamodels. Such metamodels are more flexible in approximating the response surface function, since they can capture curvature of the response surface, but on the other hand they also require a higher number of simulation runs to be conducted. The second-order regression metamodel is given by

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i \cdot x_i + \sum_{i=1}^{k} \beta_{i,i} \cdot x_i^2 + \sum_{i=1}^{k} \sum_{j=i+1}^{k} \beta_{i,j} \cdot x_i \cdot x_j + \varepsilon \tag{7}$$

where the first sum corresponds to the main effects, the second sum to pure quadratic effects, and the third sum to the interaction effects between the variables. Note from Eq. (7) that the model contains $1 + 2k + k(k-1)/2$ regression coefficients. As a result the experimental design used must contain at least this number of distinct design points and at least three levels of each design variable. The central composite design fulfils this conditions. The computation of estimates for the regression coefficients in Eq. (7) can be carried out analogously to the first-order model with OLS estimation based on Eq. (6). The substitution $\beta_{k+1} := \beta_{1,1}, ..., \beta_{2k} := \beta_{k,k}, \beta_{2k+1} := \beta_{1,2}$, ..., etc. and $x_{k+1} := x_1 \cdot x_1, ..., x_{2k} = x_k \cdot x_k, x_{2k+1} = x_1 \cdot x_2$, ..., etc. transforms Eq. (7) into the regression model of Eq. (3) with $k' = 2k + k(k-1)/2$ coded variables. The solution of Eq. (6) and back-substitution yields estimates for the regression coefficients of Eq. (7).

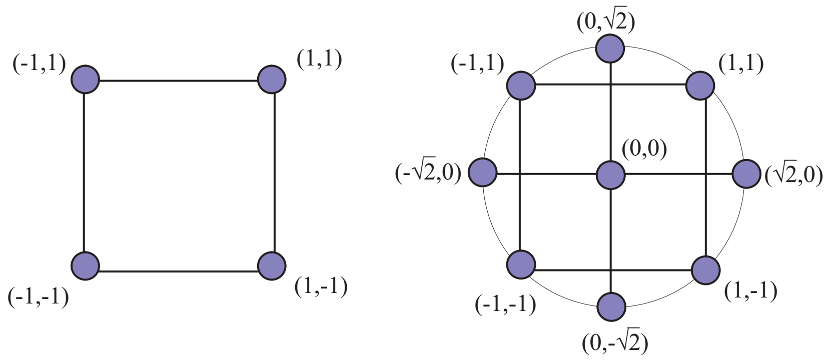

Fig. 2. 2^k factorial design (left) and central composite design (right) for k=2 factors

3.2 Testing the Regression Model for Adequate Approximation

In Section 3.1 we showed how to fit a low-order regression metamodel to the experimental data, i.e., the input parameter settings and the corresponding simulation responses. When using regression metamodels we assume that it is possible to represent the response surface adequately by a first-order or a second-order polynomial function. Thus, it should be tested if the estimated regression model adequately describes the behavior of the response in the current region of interest. For example, if we fitted a first-order model and if the response shows interaction between the factors or pure curvature, the estimated model will likely show *lack-of-fit*, which can be assessed by a statistical test.

For testing lack-of-fit, multiple observations of the response (i.e., replicated simulation runs) are needed in at least one point of the current region of interest. Suppose that we have $m_i > 1$ observations of the response for input factor values $\mathbf{x}_i = (x_{i1}, \ldots, x_{ik})$, $i=1,\ldots,n$. Let y_{ij}, denote the j-th observation of the response for input factor values $\mathbf{x}_i$. There are $m=m_1+m_2+\ldots+m_n$ observations altogether. The lack-of-fit test involves partitioning the *error (or residual) sum of squares (SS_E)*

$$SS_E = \sum_{i=1}^{n}\sum_{j=1}^{m_i}\left(y_{ij} - \hat{y}_i\right)^2 = SS_{PE} + SS_{LOF} \tag{8}$$

into a pure error sum of squares (SS_{PE}) and a sum of squares for lack of fit (SS_{LOF})

$$SS_{PE} = \sum_{i=1}^{n}\sum_{j=1}^{m_i}\left(y_{ij} - \overline{y}_i\right)^2 \tag{9}$$

$$SS_{LOF} = \sum_{i=1}^{n} m_i \cdot \left(\overline{y}_i - \hat{y}_i\right)^2 \tag{10}$$

where $\overline{y}_i$ denotes the mean simulation response (over all m_i replications) and $\hat{y}_i$ denotes the metamodels output for input factor values $\mathbf{x}_i$, respectively. As test statistic F_0 for lack-of-fit we consider (see [10]):

$$F_0 = \frac{SS_{LOF}/(n-k-1)}{SS_{PE}/(m-n)} \tag{11}$$

If the response surface function is linear, then F_0 is the realization of a $F_{n-k-1,m-n}$ distributed random variable, i.e., an F-distribution with n-k-1 nominator and m-n denominator degrees of freedom, respectively. Therefore, to test for lack-of-fit, we have to compute the test statistic F_0 and evaluate the complementary cumulative distribution function of $F_{n-k-1,m-n}$ at F_0 to get the statistical P-value. A small P-value (< 0.05) indicates, that the response surface function can not be adequately represented by a linear model, i.e., we have found lack-of-fit. If lack-of-fit is detected, we suggest to reduce the size of the region of interest or to use a higher-order regression metamodel. Nevertheless, in RSM it is not customary to fit a higher than second-order regression metamodels.

3.3 Predicting Factor Values of Improved Response

Suppose that a low-order regression metamodel is estimated according to the methods discussed in Section 3.1 and found to be an adequate approximation of the response surface function according to the lack-of-fit test presented in Section 3.2. In this section we show how to use the metamodel to derive input factors where improvement of the simulation response is expected.

First of all, we consider first-order metamodels. Since a first-order model is a planar approximation of the response surface function, the *method of steepest ascent/descent* is used to predict a direction of improved response. The direction of steepest ascent is given by the gradient of the first-order metamodel, i.e., $(\hat{\beta}_1,\ldots,\hat{\beta}_k)$, and the direction of steepest descent is given by the negative gradient, respectively. A line search is performed starting from the center point of the local region in this direction to find a point of improved response.

The question that arises is how to choose the step size for this line search. A common approach is to choose a "most important factor" x_j according to the size of its regression coefficient, i.e., $j = \arg\max_{i=1,..,k} |\hat{\beta}_i|$, and to set the step size $\Delta = 1/|\hat{\beta}_j|$, i.e., the first step results in a point on the boundary of the local region (in coded variables) corresponding to factor value x_j. Since the center of the local region in coded variables is (0,…,0), the m-th point in the direction of steepest ascent is given by

$$\left(m\Delta\hat{\beta}_1,\ldots,m\Delta\hat{\beta}_k\right). \tag{12}$$

Negation of Eq. (12) results in the line search points for steepest descent. Starting with m=1 these line search points must be transformed into natural variables and simulations must be conducted to determine the corresponding responses. To end this type of line search a stopping rule has to be chosen. The most recommended rule is to stop the line search when no further improvement of the response is observed [11]. Fig. 3 illustrates the described line search algorithm.

Next, we consider the second-order regression metamodel and assume that it is found to be an adequate approximation of the response surface. To determine a point of maximal improvement of the simulation response we use canonical analysis, i.e., the stationary point is derived from the first derivative of the regression metamodel. The fitted second-order model of Eq. (7) can be written in matrix notation as follows:

$$\hat{y} = \hat{\beta}_0 + \mathbf{x}^T \hat{\mathbf{b}} + \mathbf{x}^T \hat{\mathbf{B}} \mathbf{x} \tag{13}$$

with

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix}, \quad \hat{\mathbf{b}} = \begin{pmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_k \end{pmatrix}, \quad \hat{\mathbf{B}} = \begin{pmatrix} \hat{\beta}_{11} & \hat{\beta}_{12}/2 & \cdots & \hat{\beta}_{1k}/2 \\ & \hat{\beta}_{22} & \cdots & \hat{\beta}_{2k}/2 \\ & & \ddots & \vdots \\ \text{sym.} & & & \hat{\beta}_{kk} \end{pmatrix} \tag{14}$$

Differentiating Eq (13) with respect to **x** and setting the result equal to zero, one can solve for the stationary point:

$$\mathbf{x}_s = -\tfrac{1}{2} \cdot \hat{\mathbf{B}}^{-1} \hat{\mathbf{b}} \tag{15}$$

The nature of the stationary point can be determined by inspecting the eigenvalues of $\hat{\mathbf{B}}$. If all eigenvalues are positive (negative), then the second-order model has a minimum (maximum) at $\mathbf{x}_s$. If the eigenvalues have mixed signs, then the stationary point is a saddle point.

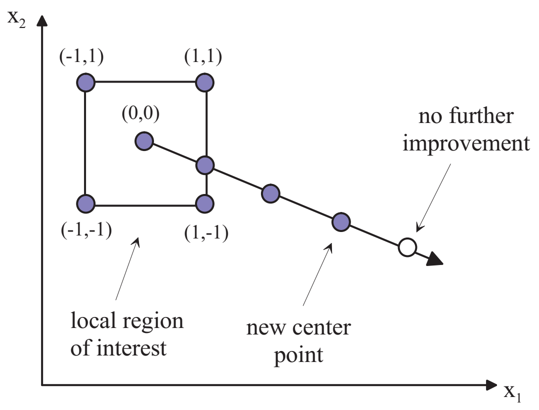

Fig. 3. Line search along the path of steepest ascent/descent

3.4 Implementation Issues

This section presents an approach that combines the building blocks of the response surface methodology as discussed in Sections 3.1 to 3.3 in a fully automated algorithmic fashion. In fact, Fig. 4 shows a high-level pseudo-code representation of the proposed RSM algorithm that we applied for the optimization of the application examples presented in section 4.

```
(1)  Transform natural variables into coded variables according to Eq. (2)
(2)  Choose initial center point $\mathbf{c}_{new} = (0,\ldots,0)$ and half-width of local region $\omega = 0.4$
(3)  WHILE $\omega > \omega_{stop}$ DO
(4)    $\mathbf{c}_{old} = \mathbf{c}_{new}$
(5)    Transform local region with center point $\mathbf{c}_{old}$ into a $[-1,1]^k$ hypercube according
       to Eq. (2)
(6)    Approximate response surface function in the local region with center point $\mathbf{c}_{old}$
       by a first-order linear regression metamodel according to Eq. (6)
(7)    Test the first-order model for lack-of-fit according to Eq. (11)
(8)    IF first-order model is adequate THEN DO
(9)      determine direction of steepest ascent/descent and step size
(10)     REPEAT
(11)       go one step in direction of steepest ascent/descent according to Eq. (12) and
           determine the response for the new factor values via a single simulation run
(12)     UNTIL new response results in no further improvement
(13)     $\mathbf{c}_{new}$ = factor values of last improvement of the response
(14)     IF $\mathbf{c}_{new} = \mathbf{c}_{old}$ THEN set new half-width $\omega := \omega/2$
(15)   ELSE DO
(16)     IF $\omega < 4\cdot\omega_{stop}$ THEN DO
(17)       Approximate response surface function in the local region with center point
           $\mathbf{c}_{old}$ by a second-order linear regression metamodel according to Eq. (6)
(18)       Test the second-order model for lack-of-fit according to Eq. (11)
(19)       IF second-order model is adequate THEN DO
(20)         Determine stationary point of second-order model according to Eq. (15)
(21)         Determine type of stationary point according to the signs of the eigen-
             values of matrix $\mathbf{B}$ in Eq. (14)
(22)         IF type of stationary point is conform with optimization goal THEN
             $\mathbf{c}_{new}$ = stationary point
(23)       OD
(24)     OD
(25)     set new half-width $\omega := \omega/2$
(26)   OD
(27) OD
(28) RETURN optimal solution $\mathbf{c}_{new}$
```

Fig. 4. Pseudo-code of the RSM optimization algorithm

When starting the algorithm one has to choose the lower and upper limits of each input parameter in order to transform the whole search space into a $[-1,1]^k$ hypercube, i.e., transform the natural into coded variables (see step (1) in Fig. 4). Furthermore, an initial center point $\mathbf{c}_{new}$ and an initial half-width ω of the local region in the response surface must be specified. If not other mentioned we assume the center point $\mathbf{c}_{new} = (0,\ldots,0)$ and local region $[-0.4,0.4]^k$ as initial values. The main steps of the algorithm are performed in the while-loop from step (3) to step (27) in Fig. 5. In our implementation we consider two types of stopping criteria, i.e., stop the RSM iteration if (i) the estimated optimal simulation response does not improve sufficiently anymore or (ii) the local region becomes too small. Note, that the pseudo-code in Fig. 4 implements the second stopping rule, where ω_{stop} is the half-width of the local region when the algorithm should stop. An implementation of the first criterion is quite similar.

In each RSM iteration the current local region is transformed into a $[-1,1]^k$ hypercube. Then the response surface function is approximated by a first-order linear regression metamodel as discussed in Section 3.1. If the first-order model is found to be an adequate approximation of the response surface function (see lack-of-fit test in Section 3.2) the line-search algorithm according to Section 3.3 is applied in order to find a point of improved response. If no improved response could be found with the line-search the half-width of local region is decreased and a new RSM iteration is started. This is exactly what is implemented in steps (4) to (14) in Fig. 4.

If the first-order model is found to be no adequate approximation of the response surface function, it is likely that the response surface has significant curvature in the local region and may be better approximated by a second-order quadratic metamodel. Nevertheless, we recommend the approximation of a second-order model only in the final steps of the optimization procedure (i.e., if ω is less than 4 times the stop-width ω_{stop}), since second-order models require much more evaluations of the simulation model than first-order models. Thus, a better strategy is to decrease the local region in order to better approximate a first-order model (see step (16) and (25) in Fig. 4).

In the final optimization phase it may be reasonable to approximate a second-order model. Similar to the first-order model a lack-of-fit test should be performed. If the approximated model shows no significant lack-of-fit, a point of improved simulation response is predicted with canonical analysis as presented in Section 3.3 (see steps (17) to (24) in Fig. 4). At the end of all RSM iterations the algorithm returns the center point of the local region, after transformation from coded to natural variables, as the optimal solution that has been found.

4 Application Examples

4.1 Optimization of a Tandem Queueing System

To illustrate the applicability of the automated RSM algorithm for optimizing simulation models, we consider a tandem queueing system as application example. This model also serves for comparison of the impact of different configurations of the RSM algorithm on the performance of the optimization process and quality of the solution.

The tandem queueing system comprises two M/M/1/K queues arranged in a row, that is, customers leaving the first queue are immediately transferred to the second queue as input customers. Both queues are assumed to have finite capacity $K = 10$. Arrivals of customers to the first queue occur according to a Poisson process with rate $\lambda = 0.5$. Each queue comprises a single server with first-come, first-served (FCFS) service discipline and exponentially distributed service time. The service rates (i.e., speed of the servers) at the first and second queue are denoted as w_1 and w_2, respectively, and are subject to be optimized by the RSM algorithm. A process chain representation of the tandem queue model is shown in Fig. 5.

The optimization problem we considered uses the following objective function that determines the revenue earned for given server speeds w_1 and w_2:

$$R(w_1, w_2) = r \cdot X(w_1, w_2) - c_1 \cdot w_1 - c_2 \cdot w_2 \tag{16}$$

where $X(w_1,w_2)$ is the throughput of the tandem queueing system (i.e., the time-averaged number of customers leaving the second queue) and r, c_1, and c_2 are constants representing a revenue factor and cost factors, respectively. In other words, a faster server is more expensive. Since $X(w_1,w_2)$ is decreasing for decreasing w_1 and/or w_2, the revenue function (16) clearly quantifies the trade-off between a high throughput (i.e., a high production rate) and costs of providing fast service.

To compute $X(w_1,w_2)$ quantitative analysis of the model must be conducted. For general models this can only be done by discrete-event simulation. Nevertheless, for the simple tandem queueing system, also numerical transient analysis of the underlying continuous-time Markov chain can be applied for its quantitative solution. For numerical analysis we applied the APNN toolbox [3]. An exact response surface function for the model, computed from transient numerical analysis at time $t = 1000$ starting with an empty system at time $t=0$ and with service rates w_1 and w_2 each varying from 0.1 to 2.0, is presented in Fig. 6. The revenue factor and the cost factors are assumed to be $r = 100$, $c_1 = 10$, and $c_2 = 30$. From this experiment we obtained the input parameter vector $\mathbf{w}_{opt}$ = ($w_{opt,1}$,$w_{opt,2}$) = (0.5959, 0.5284) that maximizes the revenue function (16). The optimal revenue found is R($\mathbf{w}_{opt}$) = 24080 EUR. Note that the revenue function of the tandem queueing system is quite sensitive for varying the service rates. In fact, a “wrong” configuration of the system may result in an unacceptable negative revenue.

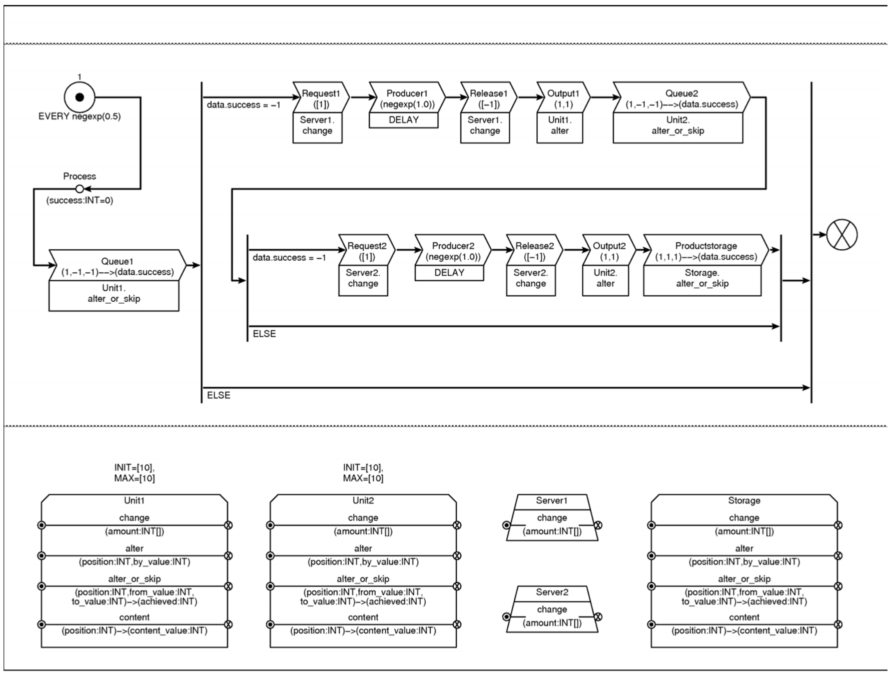
EVERY negexp(0.5)
Process
(success:INT=0)
Queue1 (1,−1,−1)−−>(data.success)
Unit1. alter_or_skip
data.success = −1
Request1 ([1])
Server1. change
Producer1 (negexp(1.0))
DELAY
Release1 ([−1])
Server1. change
Output1 (1,1)
Unit1. alter
Queue2 (1,−1,−1)−−>(data.success)
Unit2. alter_or_skip
data.success = −1
Request2 ([1])
Server2. change
Producer2 (negexp(1.0))
DELAY
Release2 ([−1])
Server2. change
Output2 (1,1)
Unit2. alter
Productstorage (1,1,1)−−>(data.success)
Storage. alter_or_skip
ELSE
ELSE
INIT=[10], MAX=[10]
Unit1
change (amount:INT[])
alter (position:INT,by_value:INT)
alter_or_skip (position:INT,from_value:INT, to_value:INT)−>(achieved:INT)
content (position:INT)−>(content_value:INT)
INIT=[10], MAX=[10]
Unit2
change (amount:INT[])
alter (position:INT,by_value:INT)
alter_or_skip (position:INT,from_value:INT, to_value:INT)−>(achieved:INT)
content (position:INT)−>(content_value:INT)
Server1
change (amount:INT[])
Server2
change (amount:INT[])
Storage
change (amount:INT[])
alter (position:INT,by_value:INT)
alter_or_skip (position:INT,from_value:INT, to_value:INT)−>(achieved:INT)
content (position:INT)−>(content_value:INT)

Fig. 5. ProC/B model of the tandem queueing system

In the following we use the optimal parameter vector $\mathbf{w}_{opt}$ as a benchmark for our response surface algorithm when evaluating the tandem queueing system via discrete-event simulation. In fact, we consider the Euclidean distance between the optimum $\mathbf{w}_{opt}$ and the best found solution of the RSM algorithm, denoted as $\hat{\mathbf{w}}_{opt}$. Recall, that the RSM algorithm can only observe the noisy surface that results from possibly replicated simulation runs and not the undisturbed response surface. In contrast to the response surface as shown in Fig. 6, Fig. 7 shows the simulation responses observed by the RSM algorithm after simulating the model for t = 1000 time units, starting at time t = 0 with an empty system.

In a first experimental setting, we compare the performance of different configurations of the RSM algorithm. In the experiments we denote

RSM(n_c, n_d) the RSM algorithms using n_c replicated evaluations of the design center point and n_d replications for the remaining design points. In all experiments the upper and lower bounds of the input parameters are l_i=0 and u_i=2, i=1,2, respectively, and the RSM algorithm starts with center point **c** = (1,1) and half-width ω = 0.4, i.e., center point (0,0) in coded variables. The RSM algorithm stops iterating if ω becomes less than 10^{-2}.

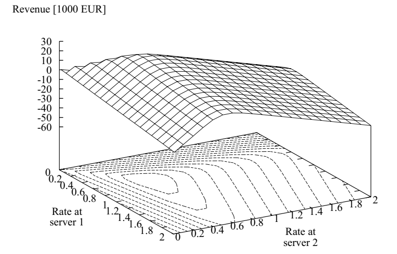

Fig. 6. Response surface for the tandem queue model

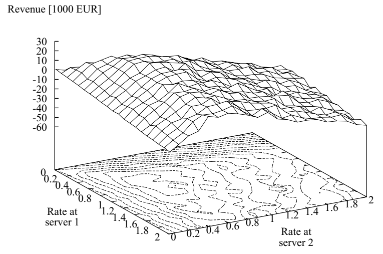

Fig. 7. Observed response surface for the tandem queue model

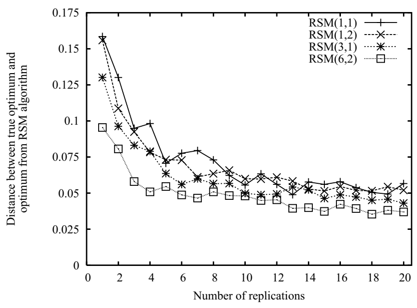

Fig. 8. Quality of found solutions for different configurations of RSM

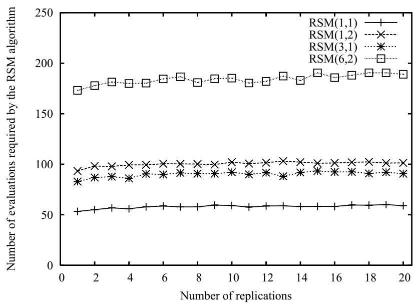

Fig. 9. Computational effort for different configurations of RSM

Recall, that the RSM algorithm calls the simulator as a black-box function. By one evaluation of the simulation model we mean one call of this black-box function. Note that one evaluation of the simulation model can comprise several replicated simulation runs, i.e., we assume that the simulator returns the mean response of (say) n simulation runs with identical input parameters but different seed values of the random number generator. Fig. 8 shows the mean distance between $\mathbf{w}_{\text{opt}}$ and $\hat{\mathbf{w}}_{\text{opt}}$ for varying the number of simulation replications that are used for one evaluation of the simulation model, where the mean distance measure is computed from 100 replications of the whole optimization procedure. Fig. 9 presents the corresponding number of evaluations required by the RSM algorithm. When increasing the number of simulation replications, the simulation responses obviously approximate the "true" response, thus, the RSM algorithm observes a less noisy surface. That is why for all RSM configurations in Fig. 9 the quality of the solution gets better for increasing the number of simulation replications.

Comparing the different configurations of the RSM algorithm, we conclude from Fig. 8 that RSM(6,2) finds the best solution and configurations RSM(1,1) and RSM(1,2) produce the worst solutions on average. On the other hand, the number of required evaluations of the RSM(6,2) strategy is much higher than for RSM(1,1) or RSM(1,2). Thus, RSM(6,2) may be not the method of choice if each simulation run is very costly, i.e., takes a long time to produce the response. Considering RSM(3,1), we observe that this strategy finds the second best solution on average but uses only about half of the evaluations of RSM(6,2). Furthermore, it is important to note, that RSM(3,1) requires less evaluations than RSM(1,2) but finds a better solution. We conclude, that RSM(3,1) should be the method of choice if simulation runs are very costly, otherwise RSM(6,2) may be more appropriate.

4.2 Optimization of a Cargo Transfer Station

As a second application example we consider a general cargo transfer station with two terminals each having five ramps. Lorries arrive to the terminals 1 and 2 with exponentially distributed interarrival times with mean 24 and 19 minutes, respectively. Arriving lorries are directed to one of the ramps. If no free ramp is available the lorry leaves the station without been unloaded. In order to unload a lorry that arrived at a ramp one of two workers is requested to come from the office to the ramp. For unloading, the worker uses a fork-lift which is located at the terminal (i.e., at each terminal one fork-lift exists). After the lorry is unloaded, the worker starts unloading the next lorry at the same terminal if at least one lorry is still waiting, otherwise he goes back to the office. On average a worker needs 2.4 minutes (drawn from an exponential distribution) to go from the office to one of the terminals as well as to go from the terminal back to the office. To unload a lorry and to transfer the load into the storage the worker needs 36 minutes on average and additional 12 minutes for cleanup after the unloading is finished. The cargo transfer station is depicted in Fig. 10.

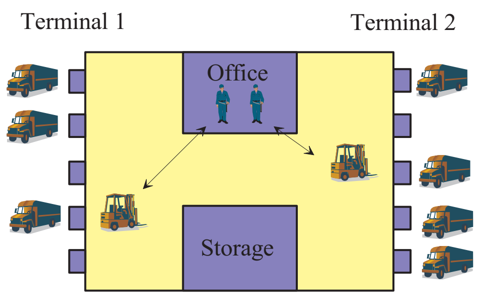

Fig. 10. Cargo transfer station with 2 terminals each having 5 ramps

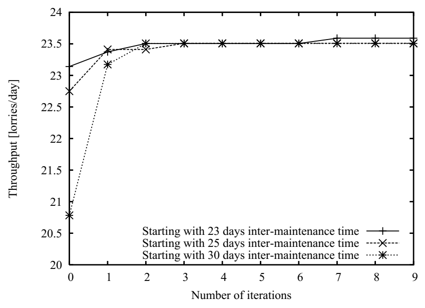

Fig. 11. Improvement of throughput per iteration during an RSM run

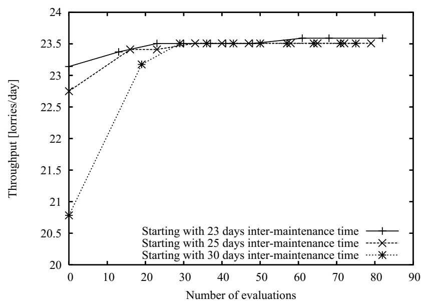

Fig. 12. Improvement of throughput per evaluation during an RSM run

The fork-lifts used in the cargo transfer station have to be maintained from time to time, otherwise they can break down due to a mechanical failure. The time to failure is assumed to be normally distributed with mean 30 working days and standard deviation of 5 days. The repair time and the maintenance time are assumed to be exponentially distributed with mean 6 days and 1 day, respectively. If a failure appears when a lorry is unloaded, the unloading is interrupted until the fork-lift is repaired. If the fork-lift is in use when it should be maintained, the unloading is completed before the maintenance is started. We assume, that a fork-lift is maintained when a deterministic time of 25 days has been elapsed after the last maintenance or repair period.

The objective of the optimization problem studied in this section is to maximize the throughput of lorries by choosing the optimal inter-maintenance interval length, i.e., the time between two maintenance periods. A small interval length reduces the probability of a failure, which would cause a long repair time, but on the other hand frequent maintenance also results in many time periods where the fork-lift is unavailable and no throughput can be generated.

For modeling and simulation of the cargo transfer station we used the ProC/B toolset and the HIT simulator introduced in Section 2. From the simulator we obtained the overall throughput of the system, i.e., the sum of the individual throughputs at each terminal. To ensure that the system is in steady-state we simulated the model for a minimum of 25 years and stopped the simulation when the throughput is within an interval of ± 8% around the mean with 90% confidence level. One simulation run takes about 15 minutes of CPU time on a Sun Sparc station with 2 GByte main memory running the operating system SunOS5.7.

Figs. 11 and 12 show the improvement of throughput when running the RSM algorithm. In particular the throughput is presented after each iteration (i.e., loop from steps (3) to (27) in Fig. 4) as well as after each evaluation of the simulation model during a run of RSM(3,1). We considered three different starting conditions as examples for the current "running state" of the system, i.e., 23 days, 25 days, and 30 days inter-maintenance time for the fork-lifts at each terminal. Starting with 30 days inter-maintenance time the RSM algorithm improves the throughput from 20.6 lorries/day to 23.5 lorries/day, which corresponds to an improvement of about 14%. Independent of the starting point the best throughput was found for inter-maintenance times of 19.5 days and 20.5 days at terminal 1 and 2, respectively. Of course, the optimum is not known for this complex model, because the complete response surface can only be generated with a huge effort. This example clearly shows the practical applicability of the proposed RSM optimization algorithm.

5 Conclusions

In this paper we present a realization of the response surface method that runs automatically and uses simulation to analyse the models. The response surface method is integrated in the ProC/B Toolset for the modeling of large logistics networks by means of process chains. By means of two examples it has been shown that the proposed approach allows the automatic optimization of complex process chains.

Although the response surface method is known for a long time, not many implementations of the method are available. The reason is that a robust implementation of the method is non-trivial and requires a lot of experience. In so far the major aspect is to optimize additional examples with the method to improve its robustness. Furthermore, we plan to extend the implemented approach by adding additional features of the response surface method such as other experimental plans and discrete parameters (see [10] for further details).

References

1. Bause F, Beilner H, Fischer M, Kemper P, Völker M (2002) The ProC/B Toolset for the Modelling and Analysis of Process Chains. Proc. 12th Int. Conf. Modelling Tools and Techniques for Computer and Communication System Performance Evaluation, London, UK, LNCS 2324, Springer, pp 51-70
2. Bause F, Beilner H, Schwenke M (2003) Semantik des ProC/B-Paradigmas. Technical Report 03001, SFB 559 – Teilprojekt M1
3. Bause F, Buchholz P, Kemper P, (1998) A Toolbox for Functional and Quantitative Analysis of DEDS. Proc. 10th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation, Palma de Mallorca, Spain, LNCS 1469, 356-359, Springer. http://ls4-www.cs.uni-dortmund.de/APNN-TOOLBOX/
4. Beilner H, Mäter J, Weißenberg N (1989) Towards a performance modeling environment: News on HIT. Proc. Int. Conf. Modelling Tools and Techniques for Computer Performance Evaluation
5. Collaborative Research Center 559 “Modelling of Large Logistic Networks”, http://www.sfb559.uni-dortmund.de/eng/index.htm
6. Fu MC (2002) Optimization for Simulation: Theory vs. Practice. Informs Journal on Computing 14: 192-215
7. Kuhn A, Hellingrath H (2002) Supply Chain Management – Optimierte Zusammenarbeit in der Wertschöpfungskette. Springer, Berlin
8. Law AM and Kelton WD (2000) Simulation Modeling and Analysis, 3rd edn. McGraw-Hill, Boston
9. Montgomery DC (2001) Design and Analysis of Experiments, 5th edn. John Wiley & Sons, New York
10. Montgomery DC, Myers RH (2002) Response Surface Methodology: Process and Product Optimization Using Designed Experiments, 2nd edn. John Wiley & Sons, New York
11. Neddermeijer HG, van Oortmarssen GJ, Piersma N, Dekker R (2000) A Framework for Response Surface Methodology for Simulation Optimization. Proc. Winter Simulation Conference, Orlando, FL, USA, pp 129-136
12. Winz G, Quint M (1997) Prozesskettenmanagement: Leitfaden für die Praxis, Verlag Praxiswissen, Dortmund