Chapter 3
Modeling and Analysis Tools

Modeling and analyzing problems from semiconductor manufacturing always require a solid knowledge of appropriate decision methods. Models are used within the production planning and control process for representing the BS and BP and for decision-making. Decision methods usually come from the areas of operations research (OR), artificial intelligence (AI), and computer science (CS). They are important prerequisites to solving decision problems.

In this chapter, we begin with a brief discussion of general systems and models. We will then describe several types of models that are used in the remainder of this monograph. Models are important to identify appropriate decision methods. We will discuss very briefly linear programming and mixed integer programming (MIP), stochastic programming, branch-and-bound techniques, dynamic programming, metaheuristics, queueing theory, and discrete-event simulation. The main ingredients of discrete-event simulation models of wafer fabs are presented in some detail.

After the development of a decision method, the question is raised of how good is the method in various situations. Therefore, we also deal with basic questions of performance assessment. We start by introducing important performance measures used in the remaining chapters of this monograph. A simulation-based method to assess the performance of a production planning and control system within a dynamic and stochastic environment is described. The content of this chapter cannot compensate for a deeper study of more specialized textbooks in OR, AI, and CS. However, in order to be as self-contained as possible, we summarize the main ideas of modeling and decision-making in this chapter.

3.1 Systems and Models

In this section, we describe how systems can be represented by models. Furthermore, we discuss different types of models.

L. Mönch et al., *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*, Operations Research/Computer Science Interfaces Series 52, DOI 10.1007/978-1-4614-4472-5_3,

3.1.1 Representation of Systems by Models

In Chap. 2, we discussed which systems and processes are related to semiconductor manufacturing. In this section, we generalize the notion of systems slightly in order to introduce the notion of models.

A system S is given by a set of components V. The components and their associations provide the structure of S. The behavior of a system is described by the interaction of the system components. An interaction is given by an information exchange or an exchange of material or energy. We call S open when it interacts with its environment. When such an interaction does not occur, the system is a closed one. An input–output system is a system that totally hides the inside view on the system components. Input–output systems are also called black-box systems (cf. Mesarović and Takahara [181]).

A real system is a certain part of the real world. The components of real systems are physical. The BS of a wafer fab described in Chap. 2 is an example of a real system. Often, we are interested only in certain aspects of a real system, and then it makes sense to work with a representation of the original system. These representations are called models.

Now, we will continue with a more abstract view of models. A model is defined formally as a triplet

$$M := (S_O, S_M, f), \tag{3.1}$$

where we denote the original system by S_O and the model system by S_M. The set of system components of S_O is denoted by V_O. The notation V_M is used for the system components of S_M. The function

$$f : V_O \to V_M \tag{3.2}$$

is called the model mapping. We are interested in models that have a high fidelity related to structure and behavior. Very often, it is not possible and even not necessary to describe f explicitly. The described situation is shown in Fig. 3.1.

There are goals that have to be achieved when creating a model. These goals are used to identify the parts of the real world that have to be modeled and interpreted in an appropriate way. The level of detail in modeling is determined by the goals. Usually, a set of model parameters have to be selected to describe the components of a model and their interactions. In doing so, it is important to remember the famous words of George Box, “All models are wrong, some are useful.” [31].

In case of a wafer fab, the type and number of machines, their characteristics, the structure of the process flows, and the job release rate are parameters of a model of the wafer fab.

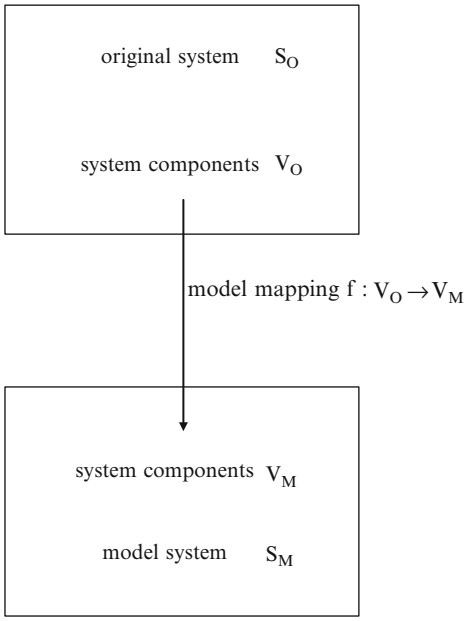

Figure 3.1: Relationship between original and model

3.1.2 Types of Models

There are different kinds of models (cf. Turban et al. [299]). Descriptive models are used to depict the components of a system and their relationships. They describe how a system behaves, but they do not explain the behavior of a system or allow for prognoses of real activities.

Prescriptive models select one or more actions among a set of alternatives. This decision is based on specified criteria. Optimization models are a typical representation of prescriptive models. In the case of these models, optimality criteria are used to select the best alternative. An optimization model consists of objective functions to be optimized and constraints that have to be satisfied. A solution of an optimization model is called feasible if it satisfies all constraints.

We also differentiate between static and dynamic models. Static models are related to a certain snapshot in time of a specific situation. Dynamic models are time-dependent. They represent scenarios that change over time. Dynamic models have the advantage that they represent the development of the system over time.

Furthermore, in the rest of this monograph, we develop deterministic and stochastic models. All model parameters are assumed to be known with certainty in a deterministic model. In contrast, a stochastic model contains certain model parameters that are described by probability distributions.

Because of the efforts to build a model and to maintain it, the simplest model that answers the question is the best. While it is generally true that prescriptive, dynamic, and stochastic models are more complicated than descriptive, static, and deterministic models, we prefer models with the latter characteristics if they are sufficient to answer the given questions. For example, a descriptive, static, and deterministic model implemented in a spreadsheet is often good enough for rough cut capacity planning.

In the next section, we discuss various decision methods that can be used in prescriptive models to select actions among alternatives. We also will describe simulation models as a main ingredient for simulation-based decision-making.

3.2 Decision Methods and Descriptive Models

In this section, we discuss various optimal and heuristic approaches that will be applied in the remaining chapters of this monograph to make decisions. Furthermore, we present some descriptive models that are useful.

3.2.1 Optimal Approaches vs. Heuristics

A decision problem consists of a set of feasible actions or sequences of actions where we have to select a particular action or sequence of actions that achieves certain objectives in the best possible way according to specified criteria. We call an action or a sequence of actions feasible when it fulfills all the requirements. The formal representation of a decision problem is called a decision model or an optimization model. In its simplest form, a decision model contains a set of alternative feasible actions and an objective function to assess them.

The notion of decision methods is closely related to decision models. Decision methods can be used to determine feasible or even optimal solutions for a certain decision problem. We introduce several decision methods in the remainder of this section in a rather generic way. Various concrete decision problems related to semiconductor manufacturing will be discussed in the remaining chapters of this monograph.

We are interested to find efficient algorithms to solve our decision problems (cf. Kleinberg and Tardos [144]). We denote by $G(n)$ an upper bound of the running time of an algorithm, where we denote by n the size of the input of the algorithm. The function $G(n)$ grows on the order $O(g(n))$ if

$$\lim_{n\to\infty} \frac{G(n)}{g(n)} = c \tag{3.3}$$

holds, where c is a positive constant and $g(n)$ is a given function. We are interested in determining $O(n^k)$ algorithms, where k is a fixed non-negative number. Algorithms of this class are called polynomial-time algorithms. If we cannot find a polynomial-time algorithm, then it is useful to check whether

it can be proved that the problem is NP-hard or not. Roughly speaking, NP-hardness means that it is unlikely that an efficient algorithm exists that will be guaranteed to solve the problem to optimality. Very often, this can be done by showing that special cases of the problem can be transformed with polynomial effort into a known NP-hard problem (see Garey and Johnson [94]).

We differentiate between optimal decision methods and heuristics. Heuristics are used to determine good solutions for NP-hard problems, but do not necessarily provide optimal solutions. Heuristics are approximation algorithms that are used to solve NP-hard problems. The majority of the decision problems discussed in this monograph are NP-hard. Therefore, we mainly focus on efficient heuristics. However, it is generally useful to know optimal decision methods, because we can exploit them to assess the performance of heuristics for small-size problem instances. On the other hand, often very efficient heuristics can be derived from combining optimal decision methods with heuristics.

In the remainder of this section, we discuss several optimal decision methods and heuristic algorithms that will later be used to tackle production planning and control problems in semiconductor manufacturing.

3.2.2 Branch-and-Bound Algorithms

A branch-and-bound algorithm is an enumerative procedure for solving discrete optimization problems optimally (see Brucker and Knust [35]). Let us consider for the sake of simplicity the following maximization problem, which can serve as a model problem.

(P) Find a feasible solution $s^* \in S$ with

$$f(s) \leq f(s^*) \tag{3.4}$$

for all $s \in S$, where we denote by S a finite set of feasible solutions and f is a real-valued objective function. Note that we can focus on maximization problems without loss of generality because we can tackle minimization problems by the same approach by simply maximizing $-f$.

The notion of subproblems is important for branch-and-bound schemes. A subproblem is a subset $S' \subseteq S$. We explain the three main ingredients of a branch-and-bound scheme as follows:

- Branching: The problem S is replaced by a set of subproblems $S_i \subseteq S$, $i = 1, \ldots, r$ with the property $\bigcup S_i = S$. This decomposition process of subproblems is called branching. The branching procedure is recursive, i.e., each subset S' can be decomposed in a similar way. We obtain a branching tree with root S and children S_i. An example for a branching tree is depicted in Fig. 3.2.
- Upper bounding: An upper bounding scheme is responsible for calculating an upper bound $UB(S')$ for the objective function value of a subproblem S'.

- Lower bounding: The objective function value of an arbitrary feasible solution $s \in S$ provides a lower bound L for problem (P). When for a subset S' the relation $UB(S') \leq L$ holds, then S' cannot provide a better solution for problem (P). Therefore, we do not need to continue the branching process from the corresponding node of the branching tree. The value of L has to be as large as possible to avoid a large number of branching steps. After some branching steps, we may reach a situation where a subproblem S'' contains only one feasible solution s. We obtain $UB(S'') = f(s)$. When $UB(S'') > L$ is valid, we replace L by $UB(S'')$.

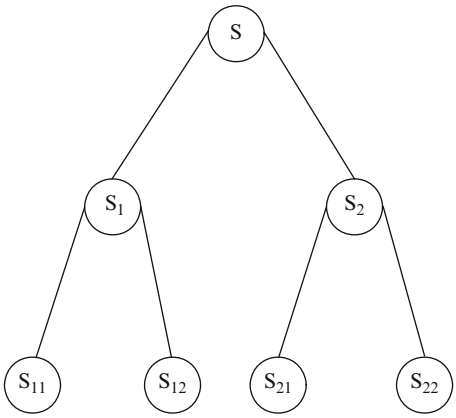

Figure 3.2: Branching tree

Branch-and-bound algorithms are important to solve small-size discrete optimization problems optimally. These known optimum values can be used to assess the performance of heuristics. When the branching tree is truncated, we obtain beam search heuristics (cf. Pinedo [240]). The truncation of the branching tree helps to reduce the computational effort.

3.2.3 Mixed Integer Programming

We introduce MIP formulations as a generalization of linear programming approaches [25, 214, 243]. A linear MIP model is an optimization model with a linear objective function that contains real- and integer-valued decision variables and linear constraints.

Each MIP model can be written in the following form:

$$Z(X) := \min_{(x,y)} \{cx + fy | (x,y) \in X\}, \tag{3.5}$$

where we denote by X the set of feasible solutions. The set of feasible solutions is described by m linear constraints, by nonnegativity constraints for

x, y, and by integer constraints for the decision variables y. Using a matrix representation, we can write the set X in the following form:

$$X := \left\{(x,y) \in \mathbb{R}^n_+ \times \mathbb{Z}^p_+ | Ax + By \geq b\right\}, \tag{3.6}$$

where

- $Z(X)$ is the optimal objective function value that is obtained by optimizing $cx + fy$ over X.
- x denotes an n dimensional column vector that contains non-negative real-valued entries $x_i, i = 1, \ldots, n$. y is a p-dimensional column vector containing integer-valued components $y_i, 1 = 1, \ldots, p$.
- $c \in \mathbb{R}^n$ and $f \in \mathbb{R}^p$ are row vectors that are given by the coefficients of the objective function (3.5).
- $b \in \mathbb{R}^m$ is the column vector of the right-hand side of the m constraints.
- A and B are $\mathbb{R}^{m \times n}$ and $\mathbb{R}^{m \times p}$ matrices, respectively.

Note that mixed binary optimization models are fairly common. In this case, $y_i \in \{0,1\}, i = 1, \ldots, p$ is valid. When $B = 0$ and $f = 0$ is true, then we have a linear optimization model that can be solved efficiently by the simplex algorithm (see Bertsimas and Tsitsiklis [25]). In the case of MIP models, branch-and-bound techniques (cf. Sect. 3.2.2) will often be applied. The main idea of these techniques for tackling MIPs consists of solving a set of linear optimization problems instead of the MIP model. When $A = 0$ and $c = 0$ are valid, we call the resulting model an integer programming (IP) model. Common software packages to solve MIPs are sophisticated and complex. The software uses a subroutine that solves linear optimization problems in the discussed branch-and-bound algorithms.

We will see in the remainder of this monograph that MIP formulations will usually be used to solve small-size problem instances for scheduling problems optimally (see Nemhauser and Wolsey [214]), whereas linear programming (LP) formulations usually can be applied for planning problems where the required level of detail is not as great.

3.2.4 Stochastic Programming

So far, we have assumed that c, f, A, B, and b in Eqs. (3.5) and (3.6) are deterministic. This assumption is often not realistic in real-world applications. In order to deal with these situations, we introduce stochastic programming as a generalization of linear and MIP (cf. Birge and Louveaux [27]). We assume that the decision model makes some decision in a first stage. Then some random events occur that affect the outcome of the first-stage decision. A recourse decision can be made in a second stage to compensate for any undesirable effects that might have been experienced as a result of the first-stage decision.

In the following, we assume for the sake of simplicity that we consider for now only linear programs, i.e., $B = 0$ and $f = 0$ in Eqs. (3.5) and (3.6),

respectively. We have to make first-stage decisions without full information on some random events. Later, full information is received on the realization of some random vector.

A two-stage stochastic linear program with fixed recourse can be written in the form

$$Z(X) := \min_x \{cx + Q(x) | x \in X\}, \tag{3.7}$$

where the set of feasible solutions is denoted by X. The set of feasible solutions is determined by m linear constraints and by nonnegativity constraints for the decision variables x. Using a matrix representation, we can write the set X in the following form:

$$X := \left\{x \in \mathbb{R}^n_+ | Ax = b\right\}, \tag{3.8}$$

where

- $Z(X)$ is the optimal objective function value that is obtained by optimizing the expression $cx + Q(x)$ over X.
- x denotes an n dimensional column vector that contains non-negative real-valued entries $x_i, i = 1, \ldots, n$.
- $c \in \mathbb{R}^n$ is a row vector that is given by the coefficients of the first term in the objective function (3.7). The function $Q(x)$ is the expected second-stage value function and will be discussed later in more detail.
- $b \in \mathbb{R}^m$ is the column vector of the right-hand side of the m constraints.
- A is an $\mathbb{R}^{m \times n}$ matrix.

The expected second-stage value function $Q(x)$ is the mathematical expectation of the second-stage value function with respect to the random vector ξ, i.e.,

$$Q(x) := E_\xi \tilde{Q}(x, \xi(\omega)), \tag{3.9}$$

where the second-stage objective function, also called the recourse function, is defined as follows:

$$\tilde{Q}(x, \xi(\omega)) := \min_y \{q(\omega) y | y \in Y\}. \tag{3.10}$$

$\tilde{Q}(x, \xi(\omega))$ is the objective function value of a second linear program with the set of feasible solutions

$$Y := \left\{y \in \mathbb{R}^p_+ | Wy = h(\omega) - T(\omega)x\right\}. \tag{3.11}$$

The quantity ω is a realization of a random variable on a probability space Ω. For a given ω, the quantities $q(\omega)$, $h(\omega)$, and $T(\omega)$ are known, where $q(\omega) \in \mathbb{R}^p$ is a row vector that corresponds to the coefficients in the objective function (3.10), $h(\omega) \in \mathbb{R}^s$ is a column vector that is the part of the right-hand side of the s constraints that do not depend on the first-stage decision variable x, and finally $T(\omega) \in \mathbb{R}^{s \times n}$ is the so-called technology matrix. $T(\omega)x \in \mathbb{R}^s$ is the part of the right-hand side of the s constraints of the set of constraints

(3.11) that depends on the first-stage decision x. Finally, $W \in \mathbb{R}^{s \times p}$ is called the recourse matrix. Putting together all the stochastic components of the second-stage data, we obtain the random row vector

$$\xi(\omega) := \left(q(\omega), h(\omega)^T, T_1(\omega)^T, \ldots, T_n(\omega)^T\right) \in \mathbb{R}^{p+s+ns}, \tag{3.12}$$

used in expression (3.9). We denote by $T_i(\omega), i = 1, \ldots, n$ the ith column of the matrix $T(\omega)$. Note that the second-stage decisions y typically are different for different realizations ω.

The representation (3.7)–(3.11) demonstrates the flow of decision-making in a two-stage stochastic program. It starts with taking first-stage decisions x in the presence of uncertainty about future realizations of ξ. Then, in a second stage, the current values of the components of the realization of ξ are known, and the recourse decision y can be made. The first-stage decisions are made in such a way that their future effects are taken into account by considering the recourse function $Q(x)$ that is the expected value of taking decision x. Note also that stochastic MIPs are useful in some situations (see Birge and Louveaux [27] for details on such optimization problems).

The following situation is important in real-world applications. We assume that there are K possible scenarios. We consider the different scenarios as realizations ω. Therefore, we have simply $\omega = 1, \ldots, K$. The probability of scenario ω is denoted by $p(\omega)$. In this case, we can calculate the second-stage value function $Q(x)$ easily. We obtain

$$Q(x) := \sum_{\omega=1}^{K} p(\omega) \left(\sum_{i=1}^{p} q(\omega)_i y_{\omega i} \right) = \sum_{\omega=1}^{K} \sum_{i=1}^{p} p(\omega) q(\omega)_i y_{\omega i}, \tag{3.13}$$

where we denote by y_ω a solution of the second-stage linear program for scenario ω, i.e., we calculate the expected value of the recourse function over all scenarios. As a result, we obtain a large linear program with a specific structure that can be solved efficiently by decomposition approaches (cf. Bertsimas and Tsitsiklis [25] for more details on such methods).

3.2.5 Dynamic Programming

Dynamic programming is another general technique to find the optimal solution of some optimization problems [35, 144]. It can be applied when the optimal solution can be determined recursively from optimal solutions of smaller subproblems. This leads to recursive formulations where the recursions are organized into stages. A dynamic programming approach generally starts with the smallest subproblems. In contrast to pure recursive algorithms, intermediate results are stored in order to avoid a repeated calculation of them.

Dynamic programming formulations are useful to obtain optimal solutions for scheduling problems, especially for single machine scheduling problems. We will see some of these applications in Chap. 5. Because of the large

computational effort and storage requirements of many optimal dynamic programming approaches, dynamic programming is often used within decomposition approaches. Based on some insights into the problem structure, the entire problem is decomposed into subproblems. Dynamic programming can be used to solve some of the subproblems.

We consider an example to illustrate this feature. A single machine batch scheduling problem in semiconductor manufacturing consists of forming batches and sequencing them. The problem can be decomposed into the subproblem of sequencing the jobs and then forming batches based on the sequence of the jobs. Dynamic programming can be used to solve the batching subproblem based on the fixed job sequence. This problem is much easier than simultaneously sequencing and batching the jobs.

Dynamic programming formulations are also often used in stochastic decision processes, especially for Markov decision processes (see Pinedo [240]).

3.2.6 Neighborhood Search Techniques and Genetic Algorithms

Next, we consider different neighborhood search techniques and genetic algorithms (GA) as examples for modern metaheuristics. A metaheuristic is a set of generic, i.e., not problem-specific, principles and schemes used to construct heuristics. Neighborhood search techniques operate on a single solution of problem (P) defined in Sect. 3.2.2 and transfer it into a new solution, while GAs maintain a population of solutions.

We start by introducing the notion of neighborhood structures. The mapping

$$N : S \to 2^S \tag{3.14}$$

is called a neighborhood structure, where we denote by 2^S the set of all subsets of S.

We introduce the notion of moves. A transformation that changes a solution s of (P) into a solution s' is called a move. The definition of a neighborhood is based on this notion of a move. We call a solution s' of problem (P) a neighbor of a solution s when s' can be obtained from s by a single move. The set of all neighbors of a given solution s forms the neighborhood of s. The following notation for the neighborhood will be used:

$$N(s) := \{s' | s' \text{ neighbor of } s\}. \tag{3.15}$$

The solution s is called the center point of the neighborhood $N(s)$. The cardinality of the set of neighbors of s is denoted by $|N(s)|$. Note that this definition of a neighborhood is clearly covered by the notion of a neighborhood structure. We can enumerate the set of neighbors by

$$N(s) := \left\{n_1(s), \ldots, n_{|N(s)|}(s)\right\}. \tag{3.16}$$

A single move

$$\mu(s, n(s)) \in N^{\mu}(s) \tag{3.17}$$

can be assigned to each neighbor $n(s)$ where we denote by $N^{\mu}(s)$ the set of possible moves. It is obvious that a neighborhood can be described by the set of moves as follows:

$$N^{\mu}(s) := \left\{ \mu_1 := \mu(s, n_1(s)), \ldots, \mu_{|N(s)|} := \mu(s, n_{|N(s)|}(s)) \right\}, \tag{3.18}$$

because we consider discrete optimization problems. Often, swapping and insertion moves are used to define neighborhoods. A swapping move exchanges two different solution elements, for example, the position of two jobs in a sequence. An insertion move removes a solution element and places it somewhere else in the solution. For example, we can remove a job from the second position in a sequence and insert it after the job that is in the fourth position.

The simplest way to design a local search algorithm is a steepest ascent-type iterative improvement algorithm for maximization problems. We start from an initial solution $s \in S$. As long as solutions $s' \in N(s)$ with $f(s) < f(s')$ exist, choose the best solution $s' \in N(s)$, set $s := s'$, and repeat this step. This algorithm terminates with some solution s^*. Generally, s^* is only a local maximum with respect to $N(s)$ because we accept only improvements of the incumbent solution.

There are different possibilities to avoid this drawback. One possibility is to restart the iterative improvement algorithm with different initial solutions. A second possibility is to explicitly take deteriorations of the objective function value into account during the iterations. When such non-improvement solutions are accepted as the incumbent solution, then it is possible to visit the same solution several times during the search process. Therefore, our neighborhood search methods may have a cyclic behavior. Consequently, cycling avoidance strategies have to be incorporated.

We discuss simulated annealing (SA) proposed by Kirkpatrick et al. [142] as a neighborhood search strategy for problem (P) that avoids cycling by selecting the current solution s' in a randomized manner. It accepts this random solution s' in iteration i only with probability

$$P(s'|s) = \begin{cases} \exp\left(-\frac{f(s)-f(s')}{t_i}\right), & \text{if } f(s) - f(s') > 0 \\ 1, & \text{otherwise} \end{cases}. \tag{3.19}$$

The numbers t_i are positive with $\lim_{i\to\infty} t_i = 0$. Often, the t_i are of the form $t_{i+1} := qt_i$, where $0 < q < 1$ is valid. When $f(s) - f(s') \leq 0$, then the move will be accepted. Because of the decreasing t_i, the probability to accept a non-improving move will be decreasing. Therefore, in later iterations, the acceptance rate for non-improvement steps will be rather small. This leads to a certain chance to get stuck in a local optimum, but the probability for this is rather small. A cycling of the solutions during the search can also be

avoided by the introduction of a tabu list. A tabu list contains in its simplest form solutions that have already been visited recently during the search. New neighbors are only accepted as an incumbent solution when they are not included in the tabu list. This type of algorithm is called a tabu search method. Usually, a solution is characterized by certain attributes that can be stored in the tabu list instead of the full representation of the solution. Typically, it is not possible to store all the visited solutions due to memory restrictions and performance issues. Therefore, we consider a list that stores only the last l visited solutions. When l is chosen big enough, then there is only a small probability of cycling. There are various tabu search variants. For a more detailed description of tabu search, the reader is referred to Glover and Laguna [102].

Variable neighborhood search (VNS) is a local-search-based metaheuristic (cf. Hansen and Mladenović [115, 188]). The main idea is to enrich a simple local-search method in order to enable it to escape local optima. This is done by restarting the local search from a randomly chosen neighbor of the incumbent solution. This restarting step is called shaking. It is performed using different neighborhood structures of increasing size. There are a couple of different VNS variants. In the following, we will briefly discuss basic VNS. For the remaining variants, we refer the reader to Hansen and Mladenović [115].

Therefore, we consider a set of neighborhood structures $N_k, k = 1, \ldots, k_{\max}$ and an initial solution $s \in S$. We choose randomly a solution s' from $N_k(s)$. Initially, we use $k = 1$. Based on s' as an initial solution, we receive a local optimum s'' by local search. When

$$f(s'') > f(s) \tag{3.20}$$

is true, then we move there, i.e., we set $s := s''$ and repeat the entire process starting from N_1. When there is no improvement obtained by s'' compared to the incumbent solution s, then we consider the next neighborhood $N_{k+1}(s)$ and repeat the shaking and the local-search step until a certain stopping criterion is meet.

VNS has the advantage that, compared to many other metaheuristics, the number of parameters to be selected is small. We only have to find a set of neighborhood structures and apply them in an appropriate sequence.

A GA maintains a set of feasible solutions called the population. GAs are motivated by principles of evolution and survival of the fittest [103, 183]. Variation operators, i.e crossover and mutation, and selection operators are used to modify the elements of the population. GAs are often appropriate for optimization problems that have the property that a combination of good partial solutions frequently leads to a good solution with respect to the objective function. A solution s is typically encoded into a sequence of symbols called a chromosome. The encoding scheme is called a representation. Each chromosome has to be evaluated by a fitness function f. The fitness function

is often based on the objective function for the optimization problem to be solved. It is well known that the performance of a GA depends strongly on the encoding scheme used (see Rothlauf [270]).

We start from an initial population of chromosomes. Each chromosome is evaluated using a problem-specific fitness function. Next, parents are selected based on their fitness. Offspring, called child chromosomes, are created from a set of parent chromosomes applying crossover operators to the parents. Crossover operators usually combine certain parts of the sequences that form the two parent chromosomes into a new child. Mutation operators are used to disorder the child chromosome with a certain probability. The child chromosomes are added to the original population. Based on the fitness that is associated with each chromosome, some chromosomes are removed based on selection operators to make sure that the size of the population is the same over all generations. The entire cycle of evaluation of the parents, offspring generation, and selection of the new population is called a generation.

We will see in Chap. 5 that GAs are useful to solve scheduling problems for single and parallel machines. But they can also be applied to solve planning problems for semiconductor supply networks or to find an appropriate mix of dispatching rules to control wafer fabs as shown in Chaps. 7 and 4, respectively.

Except for the description of stochastic programming methods in Sect. 3.2.4, so far only decision methods for deterministic problems are studied. Next, we introduce two additional methods that are related to decision-making for stochastic problems, i.e., queueing theory and discrete-event simulation.

3.2.7 Queueing Theory

The first method related to decision-making for stochastic problems is queueing theory. A simple queueing system can be characterized as an input–output system consisting of a waiting line, also called a queue, and a single server. However, there are also queueing systems that are formed by service centers and interconnecting queues. A service center consists of a number of servers working in parallel. Customers from a calling population arrive from time to time and join the queue. In some cases, the number of customers that can be waiting in the queue or in the system is limited by the capacity of the system. The customers in queue are eventually served, and after service, they leave the system. A simple queueing system is shown in Fig. 3.3.

In manufacturing, the customers of the queueing system generally correspond to jobs, and the servers are the machines. Often, the number of job arrivals to and departures from the system are of interest. Based on this information, the time that a job spends on average within the manufacturing system can be estimated. This quantity is called the cycle time (CT). Furthermore, the number of jobs within the manufacturing system that are either undergoing processing or waiting in a queue for processing can also

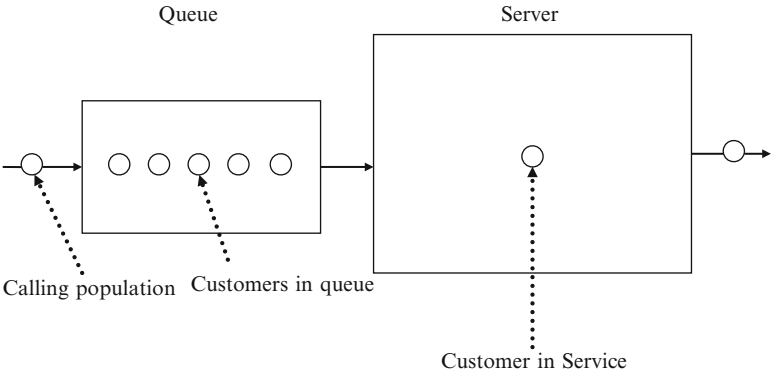

Figure 3.3: Components of a simple queueing system

be determined based on the arrival and departure information. This number of jobs is called work-in-process (WIP). The arrival process of the calling population is usually characterized in terms of inter-arrival times of successive customers. The queueing discipline determines which customer will be selected for service when a server becomes available. Common queueing disciplines include the first-in-first-out (FIFO) or last-in-first-out (LIFO) rules. Service time can be a constant or can have a random duration. Queueing theory is a mathematical approach to analyze queueing systems.

WIP and CT usually vary over time. Because it is difficult to treat this time dependency in analytic models, we are interested in time-averaged values. A queueing system is called steady state when the probability that the system is in a given state is not time-dependent. The steady-state values for WIP and CT can be considered as time-averaged values as the time becomes very large. Because we consider steady-state systems for a long time horizon, the values of WIP and CT do not depend on the initial conditions of the system. The following equation

$$\mathrm{WIP} = \lambda \mathrm{CT} \tag{3.21}$$

holds for a manufacturing system that satisfies steady-state conditions (see Little [165]). The quantity λ is the long-run input rate of the jobs to the server. Equation (3.21) is called Little's law. We consider a single server system with exponentially distributed inter-arrival times with mean rate λ and exponentially distributed service times with mean rate μ. It can be shown that the steady-state probability that n jobs are in the system, denoted by $P(N = n)$, is given by

$$P(N = n) = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^n \tag{3.22}$$

for $n = 0, 1, 2, \ldots$, where we denote by N the long-run number of jobs in this system (cf. Curry and Feldmann [55]). We use $p_n := P(N = n)$ for abbreviation. The expected WIP of the system is the expected value of a random variable that is distributed according to the discrete probability distribution (3.22). We obtain

$$\text{WIP} = E(N) = \sum_{n=1}^{\infty} n p_n = \frac{\lambda}{\mu - \lambda}. \tag{3.23}$$

It is clear that $\lambda < \mu$ has to be fulfilled to ensure a steady state for this system. Based on Little's law, we can compute the long-run CT as

$$\text{CT} = \frac{1}{\lambda - \mu} \tag{3.24}$$

because exponentially distributed inter-arrival times with mean rate λ are induced by a Poisson arrival process with mean rate λ.

When we relax the assumption of exponentially distributed inter-arrival and service times, the following approximation formula is still valid for the expected value of the time that a job spends in a queue. The corresponding random variable is denoted by T_q. The service time is modeled by the random variable T_s. We obtain for $CT_q := E(T_q)$

$$CT_q \approx \left(\frac{C_a^2 + C_s^2}{2} \right) \left(\frac{u}{1 - u} \right) E(T_s), \tag{3.25}$$

where we denote by C_a^2 and C_s^2 the squared coefficient of variation of the inter-arrival time and the squared coefficient of variation of service times, and we set $u = \lambda / \mu$ for abbreviation. The squared coefficient of variation for a random variable T is defined as $C^2(T) := \text{Var}(T)/E(T)^2$. The approximation (3.25) is called the Kingman diffusion approximation (see Hopp and Spearman [119]).

There is a notation introduced by Kendall [138] for queues. It is a five-field notation, where two consecutive entries are separated by a slash. The first entry specifies the inter-arrival time distribution, while the second entry provides information regarding the service time distribution. The third entry specifies the number of parallel servers. The maximum number of jobs allowed in the queueing system at one time is given by the fourth entry. Finally, the optional fifth entry describes the queueing discipline used. For example, M/M/1/∞/FIFO refers to a system with exponentially distributed inter-arrival and service times, a single server, an infinite capacity queue, and FIFO queueing discipline. When the fourth parameter is infinite, it is often omitted. General inter-arrival or service times are denoted by the symbol G, while the symbol M (Markovian) is used for exponentially distributed inter-arrival or service times.

In addition to the single-stage queueing model described above, it is possible to analyze the performance of queueing networks. For a more detailed introduction into queueing theory (including queueing networks) with appli-

cations to manufacturing, we refer to text books like [55, 119]. A survey that also includes a discussion of some limitations of classical queueing theory in semiconductor manufacturing can be found in Shanthikumar et al. [281].

3.2.8 Discrete-Event Simulation Techniques

Discrete-event simulation is the second method that is able to deal with stochastic decision problems. Queueing theory relies heavily on specific distributional assumptions of the underlying stochastic processes. In many real-world situations in semiconductor manufacturing, these assumptions are not fulfilled [281]. Therefore, we introduce discrete-event simulation as another important tool for decision-making in manufacturing that takes the stochastic and dynamics of the BS and the BP implicitly into account, but one that is based on less restrictive assumptions than queueing theory.

Simulation models of wafer fabs are of specific interest in the remainder of this monograph. Simulation is used to describe a process in a time-dependent manner (cf. Law [150] and Banks et al. [21]). Depending on the time progress or method, we differentiate between discrete-event and continuous simulation. In the first case, the timing of future events is determined, and the simulation jumps to the next future event. In some cases, we consider an equidistant time progression. In continuous simulation, infinitesimal small time steps are made. Continuous simulation is basically the numerical treatment of differential equations where difference equations are used to describe the changes in system variables over discrete time steps.

In this monograph, we mainly refer to discrete-event simulation. This type of simulation is based on the metaphor that entities flow through the simulation model and occupy scarce capacity that is offered by servers. There is some competition of the entities for the servers, i.e., they have to wait before they are served. In simulation models of wafer fabs, the servers are represented by resources and the moving entities by jobs.

Next, we briefly describe the main ingredients of a simulation model for wafer fabs. Depending on the goal of a simulation study, models of different complexity can be used ranging from very detailed and close to the real wafer fab to coarse and abstract. For industrial studies, very detailed models are generally used. In academic studies, both detailed and simple models are applied. A simulation model of a wafer fab can be considered as a model system S_M. Typical model components for complete wafer fab models are as follows:

- Equipment, i.e., the set of machines
- Operators and secondary resources
- Components related to material handling
- Process flows

We start by describing the JS-related (see Sect. 2.2.1) modeling issues. Apart from cluster tools, there is generally no need to model the internal behavior

of a machine, i.e., to have a detailed mechanical model or a model of the controllers that work inside a machine. Such models are used by equipment manufacturers but do not often lead to additional insight while analyzing and controlling a wafer fab. Thus, the model of the JS for wafer fabs can be kept rather simple. The typical machine-related parameters are as follows:

- Name of the machine group
- Number of machines in the machine group
- Name of the machine
- Batch-size information
- Batch formation criterion
- Setup time
- Machine qualification and dedication requirements
- Preventive maintenance cycles
- Breakdown-repair cycles

A batch machine is able to process more than one job at the same time. This number has to be specified as the number of jobs or wafers that can be batched together, and it describes the capacity of the batch machine. The batch formation criterion provides information on which jobs can be batched together. Often, we have to deal with incompatible job families, i.e., only jobs from one family can be batched together. Incompatible families are formed due to the different chemical nature (or processing time) of the different process steps on the batch machines. Sometimes, only jobs that refer to the same process step can be used to form a batch.

The setup time is the time that is required to set up a machine before processing. For a given setup state, it can be constant or depend upon the current setup state. In the latter case, we have sequence-dependent setups, and all setup times of a particular machine are listed in a setup time matrix.

Semiconductor manufacturing equipment is complex, and considerable effort is spent to keep the equipment running properly. Preventive maintenance activities are included in most wafer fab level simulation models. Often, weekly, monthly, quarterly, and yearly schedules are included. Despite efforts to prevent failures of the equipment, failures are still quite significant for many types of equipment.

One or more machine breakdown-repair cycles may exist for a machine or the entire machine group. Each cycle definition consists of a time-to-failure (TTF) and a time-to-repair (TTR) probability distribution function. In some cases, the time to failure is not counted continuously but only when the machine is busy. In other cases, failures depend upon the number of jobs or wafers processed and not upon the time-in-process state or the simulation time in general. In these latter cases, the TTF has to be given in multiples of the processing time. When there is more than one cycle for a machine, it has to be specified how to deal with parallel failures. Often, the beginning of a failure that happens when the machine is already down is postponed to the end of the current failure. Another issue concerns parallel failures on different

machines of a machine group. In particular, if the failures are used to model maintenance actions, they will generally not be performed in parallel to avoid unnecessary waiting times at the machine group buffer. The failure model has to reflect this accordingly. Problems related to the modeling of machine breakdowns in semiconductor manufacturing are described by Schömig and Rose [277].

Next, we consider the representation of operators as human decision makers in simulation models of wafer fabs. In general, due to the lack of accurate models, only simple models are applied for the humans participating in the wafer production process. In particular, interaction of operators is not modeled, i.e., the social or communication component of the human workforce is ignored. In addition to the parameters given below, operator control has to be implemented, i.e., what happens if more operators are required than are available. Usually, dispatching rules are applied in this case. The following information is required to model operators:

- Name of the operator group
- Number of operators in the group
- Skills of the operator or the operator group
- Staffing information
- Operator break cycles

The modeling of skills is important because most operators in a wafer fab are certified to run one or two machines because of the complexity of the equipment and the training costs. There are, however, a few operators who are cross-trained for multiple types of equipment. Similar to machines, operators have break cycles. There are regular breaks like lunches and random breaks like going to the bathroom. In addition, it is sometimes modeled whether operators are allowed to have breaks together or whether the breaks have to be staggered. Staffing information is required when the number of operators changes from shift to shift. It is also possible to model holidays. We refer to Mosley et al. [210] for an example of modeling operators in simulation models of wafer fabs. But often, operators are not included in simulation models of wafer fabs. This coincides with the changing role of operators in most highly automated wafer fabs (cf. the comments in Sects. 2.2.2 and 2.2.3).

Besides operators, sometimes, the detailed modeling of other auxiliary resources is necessary, especially, when these resources are scarce. Reticles in the photolithography area are an important example. The modeling of reticles within a simulation model is described, for example, in [41, 201, 228].

The main components of the MS that have to be part of the simulation model are as follows:

- Carriers
- Stockers and their assignment of bays or certain machines
- Transportation system

Note that often specialized simulation packages are used to model the JS and the MS of wafer fabs. For example, the commercial simulation packages

AutoSched AP and AutoMod are used to simulate wafer fabs. AutoSched AP simulates the JS of a wafer fab, whereas AutoMod is responsible for the simulation of the corresponding MS. The two simulation engines can be coupled by the model communication software of Brooks automation. This approach is used, for example, by Schulz et al. [278] and Pillai et al. [239] for 300-mm full-factory simulations. More simulation studies related to the MS of wafer fabs can be found, for example, in [130, 282, 316].

We continue by describing the representation of the BP within a simulation model. A process flow is required for each product manufactured in a wafer fab (see Sect. 2.2). It lists all process steps required to finish the product. In general, process flows are deterministic, but, depending on the product, it may contain alternative subprocess flows or rework loops. For each process flow, the following parameters are supplied for each process step:

- Name of the process step
- Name of the machine or machine group where the process step has to take place
- Operator requirements, i.e., the required qualification of the operators, the number of operators, and whether they have to be present during the whole period of loading, processing, or unloading or only during portions of these operations
- Auxiliary resources required
- Processing time
- Load and unload times
- Required setup state of the machine
- Amount of scrapped material
- Rework loops
- Alternative flows

The processing time is given per process step and consists of several components. Often, a machine can be loaded with more wafers than can be processed; processing of these wafers takes place in several portions. Therefore, the processing time often depends on the number of wafers or on the number of jobs. As a consequence, the processing times of jobs of the same product at a certain process step are not necessarily the same, for example, due to scrapped wafers. Note that some machines, like steppers in the photolithography work area or pipeline tools, require more sophisticated approaches to determine the processing time (see Mönch et al. [201] for the stepper case where the processing time depends on the product, on the mask level, and also on the number of ICs on a single wafer in addition to the number of wafers). The processing time for batches on diffusion furnaces in wafer fabs depends on the family of the job. The processing time associated with a batch on burn-in ovens at the back-end stage is determined by the longest processing time of one of the jobs that form the batch.

Note that in the case of cluster tools, the situation is even more complicated because the processing time depends on the sequence of processed jobs (see

Sect. 2.2 for more details on cluster tools). In principle, we have to model the internal behavior of a cluster tool to determine these processing times. Therefore, usually only simple processing time models are used in simulation models of full-size wafer fabs (see Shikalgar et al. [282]).

The load time t_{load} and unload time t_{unload} are defined as the time that is required to move a job to or from a buffer or other material handling system device before or after processing. Of course, t_{load} and t_{unload} have to add to the processing time.

At certain process steps, sometimes jobs, wafers, or dies are processed in a way that they become useless for further processing. In this case, a percentage of units scrapped is provided to reflect this behavior. This percentage is called the amount of scrapped material.

Rework is closely related to scrapped material. A percentage is given that a rework loop has to be entered either for the whole job or for several wafers. If a rework loop occurs, the whole job is processed, or a child job with rework wafers is built. The parent job may either wait until the rework loop is successfully finished and rejoin with the child job, or it may proceed immediately. At the end of the rework loop, a decision is made whether the job is allowed to continue, whether the job has to repeat the rework loop, or whether it is scrapped.

In some situations, several subflows are possible to produce different ICs that come from the same technology, i.e., a process flow consists of a sequence of subflows. For some positions in this sequence, several subflows are possible. A certain percentage is given for each of the possible subflows in this situation. At the end of the subflows, they merge again into a single process flow. It is possible that the subflow consists of a single process step. In this case, the term alternative process step is used. The concept of alternative process steps is important to model heterogeneous machines correctly.

We consider a simulation model of small complexity suggested by researchers from Intel Corporation and described by Spier and Kempf [291] as an example of a simulation model that contains typical features of a wafer fab with respect to BS and BP. It contains only three machine groups and two process flows with six process steps. The process flow is organized in two layers. Among the machine groups, there is a batch-processing machine group and a machine group with sequence-dependent setup times. The model mimics some important features of wafer fabs. We show the process flow in Fig. 3.4. We call this simulation model the MiniFab model.

The processing times of the different process steps in minutes and the maximum batch sizes in jobs are shown in Table 3.1.

Simulation studies with simulation models of full-size wafer fabs tend to be very time-consuming. This is partially caused by modeling a lot of details that are not always necessary. That is why some researchers and simulation practitioners started to consider reduced simulation models [121, 231, 263, 265, 269]. The reduction is often achieved by modeling only process steps that are related to bottleneck machines. The remaining process steps are

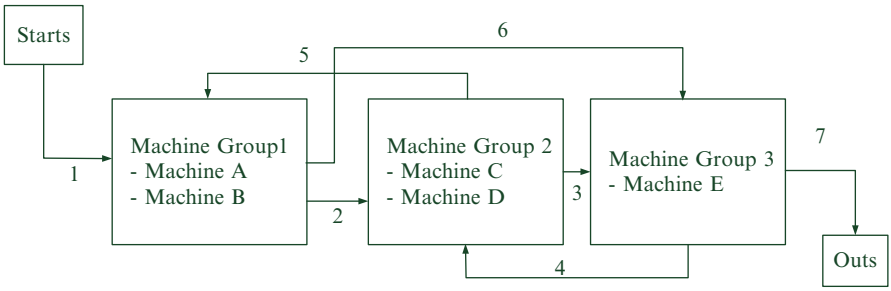

Figure 3.4: Process flow of the MiniFab model

Table 3.1: Process flows and maximum batch sizes of the MiniFab model

Process step	Machine group	Processing time	Maximum batch size
1	1	225	3
2	2	30	1
3	3	55	1
4	2	50	1
5	1	255	3
6	3	10	1

replaced by fixed time delays instead of modeling the processing of single process steps on non-bottleneck machines. However, it is far away from being trivial to find appropriate delays. Often, a large amount of simulation runs with full-size simulation models are also required to assess the quality of a reduced simulation model.

Each simulation model requires checking the logic of the model. This process is called verification. At the same time, it is also necessary to compare the results of the simulation model to reality. This step is called validation. It is usually an iterative process [21, 150] that tends to be time-consuming in the case of wafer fabs.

Therefore, reference simulation models are available for the research community to reduce the effort needed to create simulation models and to validate and verify them. Such simulation models are available in the testbed hosted at the modeling and analysis of semiconductor manufacturing (MASM) laboratory as a result of the measurement and improvement of manufacturing capacity (MIMAC) project (cf. Fowler and Robinson [83] for a description of these models that are part of the MASM Lab testbed). The MIMAC 1 model is also briefly described in Sect. 4.4.

It appears that, while simulation models are well established for single wafer fabs, it is not true for semiconductor supply networks. Only some preliminary work has been published for simulation modeling of an entire supply network (cf. Duarte et al. [74] for some recent work).

Usually, we use simulation when the analytic methods described in Sects. 3.2.2–3.2.7 do not adequately represent the system being studied. For example, in some situations, the objective function is nonlinear, some of the restrictions cannot be modeled by a set of linear constraints, or the optimization problem is simply too large to apply branch-and-bound techniques. While discrete-event simulation offers some advantage with respect to modeling capabilities, it takes considerable effort to build a simulation model, mainly because of data gathering and modeling difficulties. The main data needed to create a simulation model was described in this subsection. Experience in statistics is necessary to perform a meaningful interpretation of the simulation output.

We start by describing simulation techniques for the support of production planning and control decisions. Discrete-event simulation can be used to represent the BS and the BP. In this case, they are identified by the discrete-event simulation model. We will see that this approach is useful to assess the performance of planning and control algorithms in Sect. 3.3. In the simplest form of this approach, simulation can be used to decide the rule for which job should be processed next on an available machine. Therefore, discrete-event simulation is a tool to assess the performance of dispatching rules.

When simulation is used as a decision-making tool and not for identifying the BS and the BP, then either performance measure values as a result of planning or control instructions for BS and BP are estimated by simulation, or certain parameters that describe the behavior of the BS of interest are determined using simulation. Therefore, we differentiate between simulation-based optimization, iterative simulation, and finally the determination of immediate control instructions, i.e., which job has to be processed next on a given machine by simulation.

Simulation-based optimization starts from the idea that it is in some situations difficult to describe and evaluate an objective function (cf. Fu et al. [91] and Fu [92] for a more detailed description of the main concepts). In this situation, the objective function can be implicitly represented by a corresponding simulation model. Simulation-based optimization therefore also requires an appropriate simulation model. Optimization is typically performed using metaheuristics such as SA, GA, tabu search, or VNS. These methods tend to be computationally expensive. Therefore, the level of detail for the simulation model is important in simulation-based optimization applications. Stochastic effects typically are not neglected. They have to be taken into account both for the metaheuristic that is used for optimization and the simulation model itself. The overall scheme for simulation-based optimization applications can be described as follows.

Algorithm Simulation-Based Optimization

1. Start from a given initial solution.
2. Use the simulation model in order to estimate the objective function value by multiple runs in a stochastic setting.

3. Modify the incumbent solution by local changes using a certain metaheuristic.
4. Repeat the algorithm from step 2 onwards until a given stopping criterion is met.

Simulation-based optimization can be used either on the entire BS and BP level or for subsystems and subprocesses of it, respectively.

In contrast to simulation-based optimization, iterative simulation is used to find appropriate values for certain parameters of a decision model for planning or control problems. In production planning models, often the CT is such a parameter. Then, the production planning or control problem is solved using the current values for the parameters of interest. The resulting solution provides input for the discrete-event simulation model. For example, the quantity of jobs to be released is determined by a production planning model, then the parameter of interest is estimated using simulation. The current parameter value is updated using the simulation results. The updated value is used again as an input of the production planning or control model. The overall scheme can be summarized as follows.

Algorithm Iterative Simulation

1. Use an appropriate initial value $\rho_{\text{curr}} := \rho_{\text{init}}$ for the unknown parameter ρ. Solve the production planning or control problem using ρ_{init}.
2. Perform a simulation using the solution of the production planning and control problem as input for the simulation.
3. Determine the real parameter value ρ_{sim} as a result of the simulation.
4. Determine a new current parameter value ρ_{curr} based on the actual values for ρ_{curr} and on ρ_{sim} using, for example, exponential smoothing.
5. Solve the production planning or control problem again using the updated value for ρ_{curr}.
6. Repeat the scheme starting in step 2 until a given termination criterion is met.

Often, only a few iterations are necessary to achieve the desired maximum difference between the parameter values in two consecutive iterations. Furthermore, the initial value of the parameter of interest is often not a crucial factor. Iterative simulation in a supply chain context is discussed, for example, in Almeder et al. [7]. In this monograph, we will also see applications for setting parameters of dispatching rules due to Vepsalainen and Morton [312] in Chap. 4 and for production planning applications in Chap. 7.

Finally, discrete-event simulation can be used to determine immediate control instructions. Therefore, a simulation model that represents the current state of the BS and the BP, including a certain set of dispatching rules to decide which job has to be processed next, is necessary. An assignment of jobs to machines and sequences of jobs on single machines are determined using the dispatching rules. When different dispatching rules are used, multiple assignments and sequences are the result. Based on the preferences of a human decision maker, one specific assignment and the corresponding sequences are

chosen and then are used to make the decisions in the production control system. This concept is also known as simulation-based scheduling and will be discussed in more detail in Sect. 5.2.

3.2.9 Response Surface Methodology

Running simulation experiments is time-consuming, especially when simulation is used to find an appropriate setting of several parameter values for the BS and the BP with respect to a certain performance measure. An exhaustive search by simulation is not possible from a computational burden point of view. In this case, meta-models are an appropriate way to treat this kind of optimization problem. To model this class of problems, we assume that the performance measure value, called the response, can be expressed in the following form:

$$y := f(\xi_1, \ldots, \xi_k) + \varepsilon, \tag{3.26}$$

where the form of the true response function f is unknown and ε represents the variability. The controllable input variables are denoted by $\xi_i, i = 1, \ldots, k$. They represent the parameter values of the BS and BP that have to be varied. The term ε is treated as a statistical error. We assume that it has a normal distribution with mean zero and variance σ^2, i.e., $\varepsilon \sim N(0, \sigma^2)$. The variables ξ_i are called natural variables. However, it is often more convenient to work with dimensionless variables that are normalized, i.e., from $[-1, 1]$. These variables are called coded variables and can be achieved by some transformation of the natural variables. The true response function can be represented in the form $\eta := f(x_1, \ldots, x_k)$. Because f is unknown and often complicated, we approximate it by a low-degree, usually first- or second-order polynomial. Second-order meta-models are of the general form

$$y := \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \beta_{ii} x_i^2 + \sum_{j} \sum_{i \neq j} \beta_{ij} x_i x_j + \varepsilon, \tag{3.27}$$

where the unknown coefficients β_0 and $\beta_i, \beta_{ij}, i = 1, \ldots, k, j = 1, \ldots, k$ are called the parameters of the model that have to be fitted from results of simulation runs. Note that we obtain a first-order model out of the second-order model (3.27) when we select $\beta_{ii} = 0$ and $\beta_{ij} = 0$. While least square analysis is often used to estimate the significant parameters for the meta-model, analysis of variance techniques is used to determine which parameters are statistically significant.

We assume that the discrete-event simulation model is a reasonable representation of the BS and the BP. If the meta-model is an adequate approximation of the corresponding simulation model, then the optimization of this model will be approximately equivalent to the optimization of the BS and BP. Therefore, after we have determined the second-order meta-model, we can use it instead of the simulation model to optimize the response by

determining appropriate values for the coded variables. By using standard optimization techniques, i.e., steepest descent for the first-order model and conjugate gradient methods for the second-order meta-model, the corresponding optimization problems can be solved using standard optimization software.

For a more detailed introduction into the response surface methodology, we refer to Myers et al. [212]. Some applications of meta-modeling techniques in semiconductor manufacturing are shown by McAllister et al. [177]. Chapter 4 contains applications related to the design of blended dispatching rules in semiconductor manufacturing.

3.2.10 Learning Approaches

Following Russell and Norvig [273], a basic learning system contains a performance element and a learning element. The performance element decides what actions are to be taken, whereas the learning element modifies the performance element to allow it to make better decisions. The learning element takes feedback from the environment into account during learning. A reasoning mechanism is responsible for the learning.

In a certain sense, the regression analysis described in Sect. 3.2.9 might be considered to be learning a continuous function from examples of its input and output. The resulting meta-model is the performance element, while least square analysis is used as the reasoning mechanism.

Next, we continue with a discussion of neural networks as a widely used learning approach. A neural network consists of a set of nodes, called neurons, and the neurons are connected in different ways. Each neuron is used to implement an activation function T:

$$y_i := T\left(\sum_{j=1}^{n} w_{ji} u_j\right). \tag{3.28}$$

This function determines output values as a result on given input values. Input values are denoted by $u_j, j = 1,\ldots,n$. The output values are denoted by $y_i, i = 1,\ldots,m$. The quantities $w_{ji}, j = 1,\ldots,n, i = 1,\ldots,m$ are the corresponding weights. The neurons are organized in layers. Besides input and output layers, a neural network also has internal layers. Input values are transformed into output values using the weighted input values.

Applying neural networks to make a certain decision requires determining appropriate input values to characterize the problem and to represent the solution of the problem by the output values. Training data, i.e., a certain set of problem instances with known high-quality solutions, also called examples, is used to discover how the output values depend on the input values. It is clear that the neural network as a meta-model is the performance element, while the learning approach for the weights is the learning element.

The main advantage of neural networks is their ability to learn by adapting the weights. Furthermore, neural networks are able to deal with noisy

data. Nonlinear functions can be represented by neural networks. The basic limitation of the neural network approach is that a new network has to be constructed when the underlying situation has changed. This often requires a large training effort. At the same time, training data is often not available in case of a dynamic and stochastic BS and BP. In Chap. 5, we will mention applications of neural networks in scheduling.

Inductive decision trees are another learning approach. A decision tree takes, as input, a situation described by a set of attributes and returns the predicted output value for the input. A decision tree makes decisions by performing a sequence of tests. The tree consists of internal nodes and leaf nodes. The leaf nodes represent the output value that is the result when this leaf is reached. The internal nodes test the value of one of the properties. The learning is performed by using a set of examples, i.e., a mapping between situations and output values to determine the decision tree (see Russel and Norvig [273]).

3.2.11 Summary of Decision Methods and Descriptive Models

Finally, we would like to combine the different type of models presented in Sect. 3.1.2 and the various methods from Sect. 3.2. The resulting scheme is shown in Fig. 3.5. We can clearly see that while discrete-event simulation is both descriptive and prescriptive at the same time, it is always dynamic. Only dynamic programming and discrete-event simulation are deterministic and stochastic. Queueing theory is descriptive and contains elements of both static and dynamic models, but it is stochastic. The response surface methodology is descriptive and can be also prescriptive.

It is not reasonable to assign the learning approaches discussed in Sect. 3.2.10 to any of the model types from Sect. 3.1.2 because learning models are often used to support the parameter selection process for decision methods. Therefore, we avoid a specific assignment.

It is interesting to remark that, as already stated in Sect. 3.1.2, dynamic and stochastic models are difficult to analyze. We can see from Fig. 3.5 that only discrete-event simulation is able to deal with these kinds of models. Based on this insight, we can conclude that discrete-event simulation is of particular importance.

3.3 Performance Assessment

In this section, we present a performance assessment methodology. In addition, we discuss an architecture that allows for simulation-based performance assessment of production planning and control approaches.

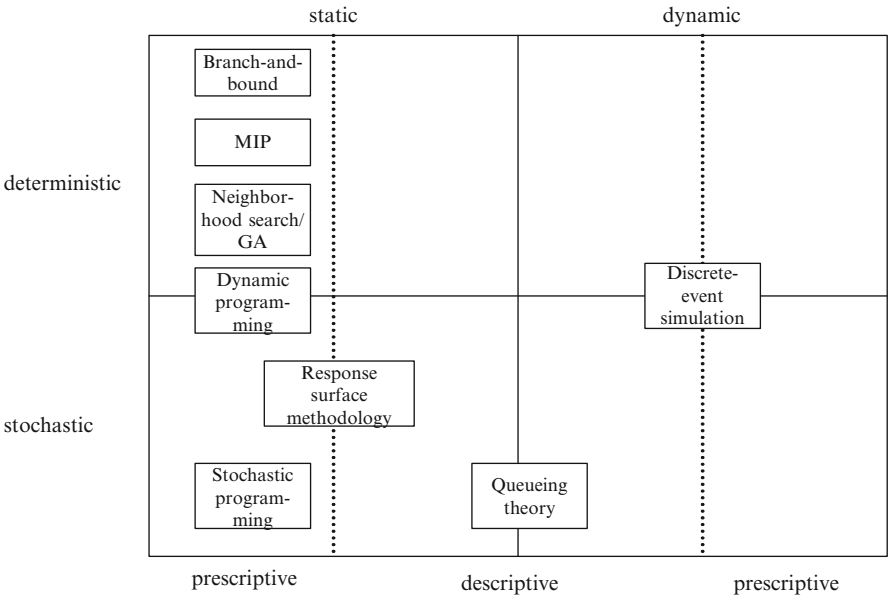

Figure 3.5: Scheme for decision methods and descriptive models

3.3.1 Performance Assessment Methodology

When a new decision-making algorithm is proposed, in the beginning, its performance is unknown. Therefore, it is necessary to assess its likely performance before it can be applied in a wafer fab. There are, in principle, two possibilities. The first one takes a snapshot of the state of the BS and the BP and then determines the values of certain performance measures. This type of performance assessment is described in some detail in Rardin and Uzsoy [257]. The second one repeats running the decision-making algorithm in a time-based or event-based manner taking the current state of the BS and the BP into account. We refer to the first approach as a single instance-based performance assessment, whereas the second one is called a rolling horizon-type performance assessment scheme. Note that the single instance-based approach is rather static, while the latter one is much more dynamic and takes appropriate feedback of the BP and BS into account.

In a next step, we describe a performance assessment methodology that can be applied in both situations. In this methodology, a fixed production planning or control approach with unknown performance is considered. In order to assess its performance, the following scheme is suggested:

- Determination and specification of the appropriate performance measures
- Determination of production planning and control approaches used for comparison with the new approach
- Description of different problem instances and simulation scenarios in the first and second cases, respectively, and specification of designed experiments
- Specification of the performance assessment strategy
- Description of the hardware and software environment for the performance assessment
- Running the new planning and control approach for each problem instance or, in a rolling horizon manner, by discrete-event simulation for each simulation scenario in the first and second cases, respectively, and discussion and interpretation of the results

We start by describing the difference between direct and indirect performance measures. This distinction corresponds to the planning and control setting presented in Fig. 2.2. Direct performance measures consider quantities that have direct influence on the performance of the manufacturing system related to the BS and the BP. Throughput (TP), CT (cf. Sect. 3.2.7), and total weighted tardiness (TWT) are examples of direct performance measures. TP is defined as the number of completed jobs leaving a system within a certain period of time. The TP rate is consequently the number of completed jobs per unit of time. In the MS, the number of moves of the vehicles is a TP-related measure. Closely related to TP is the makespan $C_{\max}$ of a set of n jobs. It is defined by the expression

$$C_{\max} = \max\left\{C_j | j = 1, \ldots, n\right\}, \tag{3.29}$$

where we denote by C_j the completion time of job j. The measure CT for job j is defined as

$$\mathrm{CT}_j := C_j - r_j, \tag{3.30}$$

where r_j denotes the ready (release) time of j. The average CT (ACT) of n jobs is given by

$$\mathrm{ACT} := \frac{1}{n} \sum_{j=1}^{n} (C_j - r_j). \tag{3.31}$$

ACT is an important measure in semiconductor manufacturing because a small ACT value may lead to large yield because the probability of contamination on the shop floor is smaller in case of a small CT (see Atherton and Atherton [14]). The carrier delivery time (CDT) is the counterpart of the BS-related CT within the MS. In some situations, the variance of CT is of interest. It is defined by

$$\mathrm{Var}(\mathrm{CT}) := E((\mathrm{CT} - E(\mathrm{CT}))^2) \approx \frac{1}{n-1} \sum_{j=1}^{n} (\mathrm{CT}_j - \mathrm{ACT})^2. \tag{3.32}$$

Closely related to CT is the measure total flow time or total completion time (TC). It is defined as

$$\mathrm{TC} := \sum_{j=1}^{n} C_j = n\mathrm{ACT} + \sum_{j=1}^{n} r_j, \tag{3.33}$$

i.e., minimizing ACT is equivalent to minimizing TC. Its weighted counterpart is the total weighted completion time:

$$\mathrm{TWC} := \sum_{j=1}^{n} w_j C_j, \tag{3.34}$$

where we denote by w_j the weight of job j. On-time delivery performance-related measures are also important because of possible customer satisfaction and hence advantage in the fierce competitive market. The tardiness T_j of a job j is given by

$$T_j := \max(C_j - d_j, 0), \tag{3.35}$$

whereas the performance measure TWT is defined by

$$\mathrm{TWT} := \sum_{j=1}^{n} w_j T_j, \tag{3.36}$$

where we denote by d_j the due date of job j. The average weighted tardiness (AWT) is defined as

$$\mathrm{AWT} := \frac{1}{n}\mathrm{TWT}. \tag{3.37}$$

When we have $w_j \equiv 1$, then we are talking about total tardiness (TT). We use the notation AT for the average tardiness.

Somewhat related to TT is the maximum lateness. This performance measure is defined as follows:

$$L_{\max} := \max\{L_j | j = 1, \ldots, n\}, \tag{3.38}$$

where we denote by $L_j := C_j - d_j$ the lateness of j. Similar to Var(CT), the variance of the lateness $\mathrm{Var}(L)$ is defined by

$$\mathrm{Var}(L) := E((L - E(L))^2) \approx \frac{1}{n-1}\sum_{j=1}^{n}(L_j - \mathrm{AL})^2, \tag{3.39}$$

where the average lateness (AL) is given by $\mathrm{AL} := 1/n\sum_{j=1}^{n} L_j$.

Finally, in some situations, the number of tardy jobs (NTJ) is of interest. This performance measure is defined as follows:

$$\mathrm{NTJ} := \sum_{j=1}^{n} U_j, \tag{3.40}$$

where we have

$$U_j := \begin{cases} 1, \text{ if } 0 < C_j - d_j \text{ for job } j \\ 0, \text{ otherwise.} \end{cases} \tag{3.41}$$

The weighted counterpart of NTJ, the weighted number of tardy jobs (WNTJ)

$$\text{WNTJ} := \sum_{j=1}^{n} w_j U_j, \tag{3.42}$$

is also considered. TT, TWT, $L_{\max}$, NTJ, and WNTJ are important measures in terms of on-time delivery performance. In Table 3.2, more examples for direct performance measures are shown.

Table 3.2: Examples for direct performance measures

Class	Example
due date-oriented	maximum lateness $L_{max} := \max\{C_j - d_j \mid j = 1, \ldots, n\}$
throughput-oriented	TP of a certain work area
cycle time-oriented	ACT of the jobs of a certain product family
load-oriented	WIP

Note that in many practical situations, we have to deal with multiple criteria that are sometimes in conflict to each other. The desirability function approach in optimizing multiple criteria of interest was originally suggested by Derringer and Suich [66]. The approach transforms each objective value into a value between 0 and 1. Thus, each criterion y_i is converted into an individual desirability function d_i that varies over the range zero to one. If y_i is outside the acceptable range defined by the user, then $d_i = 0$. However, if y_i meets the goal, then $d_i = 1$. Let U_i be the maximum allowable value for the response y_i, and let G_i be the goal value for y_i. We define d_i as follows:

$$d_i := \begin{cases} 1, & \text{if } \; y_i < G_i \\ \left((U_i - y_i)/(U_i - G_i)\right)^{z_i}, & \text{if } \; G_i \le y_i \le U_i, \\ 0, & \text{otherwise} \end{cases} \tag{3.43}$$

where $z_i > 0$ is a real number known as the weight on the desirability function. When $z_i = 1$ for each objective i, the desirability function is linear. Choosing $z_i > 1$ places more emphasis on being close to the goal value, while setting $0 < z_i < 1$ decreases importance on proximity to the goal value. Once the individual desirabilities have been calculated, the combined desirability D that is to be maximized is computed as the geometric mean of the individual desirabilities. We obtain for the case of m desirabilities:

$$D := \left(\prod_{i=1}^{m} d_i \right)^{1/m}. \tag{3.44}$$

We will see applications of the desirability function approach related to dispatching and scheduling in Chaps. 4 and 5, respectively.

In contrast to direct performance measures, indirect performance measures are not directly related to the performance of the manufacturing system. They are used to evaluate properties of certain production planning and control algorithms. Consequently, they are related to the PS, PP, CS, and finally the CP. Robustness and stability measures are examples for this class of performance measures.

Performance measures related to robustness measure the change in the objective value of a plan or schedule after plan or schedule revisions. Stability of a plan or schedule is defined as the deviation of the final plan or schedule related to the original plan or schedule (cf. [106, 141, 233]). The deviation of two schedules can be measured, for example, as the difference of the completion times of jobs. Stability measures the difference of initial and of executed plans and schedules under the influence of disruptions. More indirect performance measures are shown in Table 3.3.

Table 3.3: Examples for indirect performance measures

Class	Example
re-planning effort-oriented	number of required re-planning activities
run time-oriented	run time of a certain production planning algorithm
agility	time needed to obtain the original WIP after the breakdown of a major bottleneck machine
stability	deviation of the final plan or schedule from the original one

Next, we have to discuss production planning and control approaches for comparison with the new approach. Decentralized and centralized production control approaches can be distinguished. Hierarchical approaches are somewhere between these approaches. Decentralized approaches work on local data, and coordination and cooperation issues become important in order to avoid the limitation of the myopic view of decentralized approaches. Dispatching rules in semiconductor manufacturing are an example of this type of approach (cf. Chap. 4 for more details on dispatching rules). Centralized approaches are given, for example, by LP, MIP, and by various kinds of neighborhood search techniques.

Often, it makes sense to use exact procedures like branch-and-bound or MIP described in Sect. 3.2 for small-size problem instances to ensure the correctness of implementations and to get a sense of the performance of the new approach. For large-size problem instances, heuristics are used for comparison. In some situations, it is possible to compare the performance of the new approach with the approach that is used in the company by considering problem instances or simulation models and the corresponding production planning and control instructions from the company.

We describe now the determination of problem instances and simulation scenarios. We start with the generation of synthetic problem instances. In the case of individual problem instances, we determine a set of factors that we expect to have an impact on the performance of the new algorithm. We consider levels for each factor, i.e., we vary the values of the factors in a controlled way. Often, the levels are selected as realizations of a random variable that follows a prescribed probability distribution. In the case of full factorial experiments, problem instances are generated for each factor combination. Usually, a certain number of stochastically independent problem instances are generated for each factor combination. Methods from the theory of designed experiments can be used to study the impact and the effect of different factors (cf. Montgomery [208]). One of the primary approaches to designing an experiment is to select the location of design points to optimize some criterion function; this is often called optimal design of experiments (cf. Pukelsheim [249]). Typically, this criterion is related to the variance/covariance matrix of the parameters in the model. The most popular optimization criterion is D-optimality, which seeks to minimize the volume of the joint confidence region of all the parameters in the model.

It is also possible to avoid the generation of problem instances by taking problem instances directly from the BP. Alternatively, for certain classes of problems, benchmark instances are publicly available (cf. the OR-Library [220] for a collection of such problem instances available over the web).

Performance assessment experiments for rolling horizon approaches are organized as different simulation scenarios. In contrast to the problem instance case, usually the number of scenarios is fairly small. A single simulation scenario is given by a set of independent parameters, a range of variation for the parameters, and a set of dependent variables, i.e., basically the performance measure values. Each scenario is represented by a specific simulation model. By using designed experiments (see Montgomery [208]) and meta-modeling techniques like response surface methodology (see Myers et al. [212] and Sect. 3.2.9), it is possible to reduce the number of required simulation runs by constructing an appropriate meta-model, often a regression model, and optimizing the performance measure values using the simpler meta-model instead of the simulation model. Furthermore, modern variance reduction techniques can also help to decrease the number of necessary simulation runs (cf. McAllister et al. [177]). Often, the simulation scenarios are based on data that is collected in a wafer fab. But at the same time, the scenarios are often also based on reference simulation models like the MASM Lab testbed (see also the description in Sect. 3.2.8). The usage of such public benchmark models offers some additional advantage as the results are now comparable because they are company-independent.

The choice of the performance assessment strategy is important because of the stochastic behavior of manufacturing systems. There are two main possibilities to assess the performance of a system in simulation modeling.

In the first case, the stationary behavior of a complex manufacturing system is of interest. After the warm-up period, the performance indicators of interest are measured. Long runs have to be performed. However, because of the stationary behavior, a relatively small number of independent runs is often enough (see Law [150]). The performance measure values are taken as the average of the performance measure values for the single runs. Besides average values that estimate the mean, confidence intervals are provided to estimate the range of values, which is likely to include the unknown performance measure value.

Looking at the transient behavior of a manufacturing system is the second possibility. Transient behavior appears during the transition phase from one system state to another. Product mix changes, ramp-up, and introduction of new machines lead to transient behavior of the manufacturing system. Studying a manufacturing system in the transition phase requires the measurement of the performance measures of interest in a time-dependent manner. As opposed to the stationary case, a considerably larger number of simulation runs is required in order to obtain statistically significant results for the performance measures. The length of a single run is determined by the length of the transition phase. The performance measure values are calculated as the average of the performance measure values obtained at a single point of time.

A performance assessment strategy is basically given by the decision about whether to investigate a stationary manufacturing system or a manufacturing system in the transient phase from a given stationary to another stationary behavior or not. Furthermore, the number of replications and the run length are also part of the performance assessment strategy. In the case of single problem instances, the maximum amount of computing time for each problem instance or the number of replications of computations are also elements of a performance assessment strategy.

In order to compare different production planning and control algorithms from a run time behavior point of view, it is required to fix a hardware and software environment for the performance assessment. It is furthermore required to describe the used load balancing strategy of the computer network in the case of distributed production planning and control algorithms. In the final step of the performance assessment scheme, it is required to run the experiments for all problem instances or all simulation scenarios. Then, the results have to be analyzed and interpreted. Based on the results, often several refinements of the design of experiments are necessary.

We note that additional performance measures that are related to the performance of the BS and the BP of wafer fabs can be found in Leachman and Hodges [155].

3.3.2 Architecture for Simulation-Based Performance Assessment

We continue by describing a simulation-based architecture that allows for simulation-based performance assessment of production planning and control approaches. There are two principle possibilities to incorporate production planning and control algorithms into a discrete-event simulation tool. In the first possibility, the production planning and control algorithm basically uses the information and data on the level of the simulation tool. As a result of this strategy, proprietary source code is obtained that is difficult to understand and to maintain. It is rather complicated to implement different production planning and control approaches. However, for comparison purposes, the ability to plug in different production planning and control approaches is highly desirable.

The second strategy is much more flexible. A blackboard-type data layer (see Mönch et al. [202]) that acts as a mirror of the base process BP emulated by the simulation model is used. The simulation tool is responsible for an update of the corresponding objects of the data layer in the case of the occurrence of well-defined events. When, for example, a job is released to the shop floor, i.e., it is started in the simulation, then a corresponding job object is created in the data layer. Other events of importance are, for example, the start of a setup operation of a certain machine or the completion of a certain process step, i.e., the job leaves a machine. In the latter case, the state of the job object is changed in the data layer, whereas in the first case, the state of the machine object is changed. Besides business objects like machines, jobs, and products, the data layer contains objects that represent the production planning instructions *mp* and production control instructions *mc*. Furthermore, it contains objects that are used to store statistics. The simulation engine implements the production control instructions *mc* in a dispatching manner.

The data layer is located in the memory of the computer; hence, fast access is possible. The incremental, event-driven update of the business objects avoids time-consuming queries from databases. The object model of the data layer is much easier to understand and to maintain than the proprietary data structures of a certain simulation tool. When the BS is segmented into different work areas, a separate blackboard-type data layer can be assigned to each segment of the manufacturing system. The architecture has to contain a persistency mechanism that supports object state and performance measure tracing. Using object-oriented databases seems to be appropriate for that purpose because of the highly nested objects from the data layer. The objects of the data layer are stored in a periodic manner.

The built-in mechanism for job selection and machine load of the discrete-event simulation tool acts as a dispatcher in a natural way. The dispatcher uses sorted lists with jobs, so-called dispatching lists, calculated using production control algorithms. When a machine becomes available in the simulation, the next job to be processed is determined from the dispatching list

of the machine. Furthermore, the simulation tool also specifies under what circumstances the production control algorithm is applied to calculate new dispatching lists. This feature is denoted as logic for calling of the production control approach.

The architecture is completed by a demand forecast module and by a demand generation module. While the first module is designated to the determination of forecast using historical demand, the second one generates concrete demand that is used as an important input for the planning algorithms of the PS. The PS determines which quantities have to be released in a certain point of time into the BS. This information is used by the CS to calculate the *mc*. The architecture is shown in Fig. 3.6.

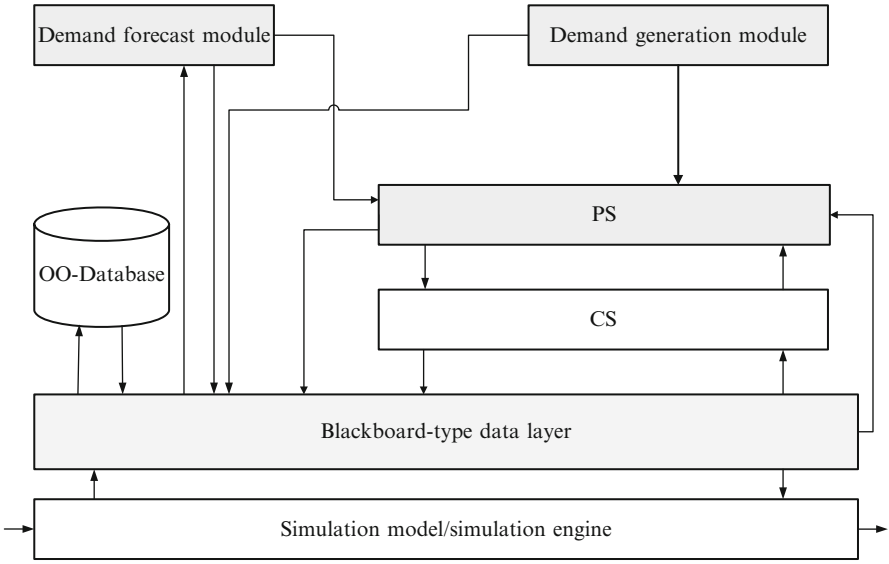

Figure 3.6: Simulation-based architecture for performance assessment

Building production planning and control applications from scratch is not very common because of available commercial and academic optimization and scheduling libraries and of available editors for dispatching rules. Examples are the ILOG solver and scheduler classes (cf. Le Pape [152]), class libraries for GAs (cf. Pain and Reeves [227]), and the dispatching rule editor from the simulation tool AutoSched AP. Therefore, an architecture for benchmarking has to take this fact into account. Because most of the available class libraries are written in the C++ programming language, the degree of freedom for choosing an appropriate implementation language for the architecture is limited. The suggested architecture allows for a plug-in of different production planning and control approaches.

In the described architecture, we only model the JS. However, an extension of this architecture that also includes the MS of a wafer fab is described by Driessel and Mönch [71]. In this situation, we have to use one simulation model for the JS and a second one for the MS. Note that we will provide certain examples for the usage of the simulation-based architecture for performance assessment of scheduling and production planning approaches in the remaining chapters of this monograph.