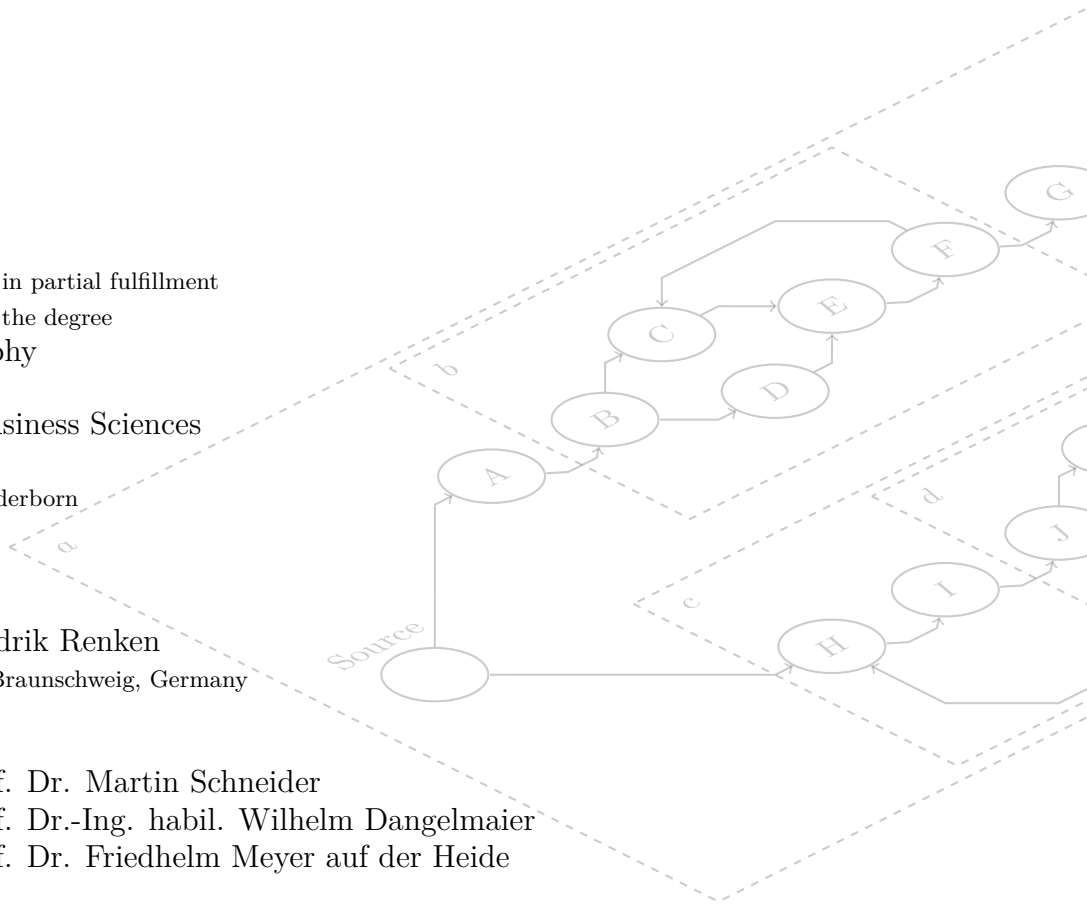
Acceleration of Material Flow Simulations

Using Model Coarsening by Token Sampling and
Online Error Estimation and Accumulation Controlling



Dissertation submitted in partial fulfillment
of the requirements for the degree
Doctor of Philosophy
in the subject of
Economics and Business Sciences
(Dr. rer. pol.)
at the University of Paderborn

By
Dipl.-Inform. Hendrik Renken
born on 19.04.1981 in Braunschweig, Germany

DEAN	Prof. Dr. Martin Schneider
REFEREE	Prof. Dr.-Ing. habil. Wilhelm Dangelmaier
CO-REFEREE	Prof. Dr. Friedhelm Meyer auf der Heide

November 6, 2013

Created at
University of Paderborn
Heinz Nixdorf Institute
Businesscomputing, esp. CIM
Prof. Dr.-Ing. habil. Wilhelm Dangelmaier
Fürstenallee 11
33102 Paderborn

I would especially like to thank
my wife, Irina,
Sascha Brandt and
Maureen Winter
for their continuing support
while creating this document.

"And, bonus, it's a university press book, which means
it contains things like *facts* and *information*." - Zach Weiner

Contents

Chapter 1

Introduction

Today's production planning faces major challenges: The life cycle of new products gets noticeably shortened while development and production costs rise. This is caused by the ever increasing complexity of the products and their production. While they are getting a smaller and lighter design, new features have to be integrated. This requires new and hard to handle materials and increasingly complex production systems. To be most competitive on the world market, these systems must be well understood and controlled. Only then production overhead and waste and therefore production costs can be minimized.

Simulation offers an extensive degree of understanding, verification and controlling of production systems. It is used to study the behavior of a system in certain situations. From the observed behavior, knowledge about the system's inner workings can be gained. This knowledge can then be used to make decisions on how to handle and control the system under study. Typically, the simulation process begins with the creation of a system model that can be studied instead of an actual system. This approach is necessary when a system is unavailable, the system is too expensive, or the overall situation could prove dangerous.

Especially the simulation of material flow models supports decision making regarding the production planning. Simulation can help to understand a systems behavior in critical situations in depth. This knowledge helps safeguard the production. Furthermore, material flow simulation is used to optimize production processes, layout- and workload planning. However, such a utilization of simulation requires detailed production system models. While current models are already very detailed (often covering the whole company), the desire for even more detailed simulation models continues. Even with the ever increasing computing power and easier-to-use simulation

software, this hunger for more detailed and widespread simulation models cannot be satisfied [CBP00].

To solve this problem, the amount of computational resources can be increased, e.g. by parallelizing the simulation. Another option is the reduction of the complexity of the simulation model. This reduction process, often called *abstraction* is normally done by the modeling engineer who must have a certain degree of experience to create valid reduced models [BT00]. A model is said to be *valid* when the decisions derived from the observation of the simulated model match the decisions derived from the original system's behavior. Because the work of such an engineer is very expensive in terms of time and monetary costs several automated approaches have been proposed. These approaches take a simulation model and analyze its behavior under specific conditions. Based on this analysis certain parts of the model, or even the whole model, are replaced with new components. These components usually have a similar behavior as the original model part but need less time to compute.

Current coarsening and simplification approaches suffer from several drawbacks: For large or complex models, analytical methods are not applicable. In some related approaches, the coarsening/simplification step still has to be done by an experienced modeler. Most of the automated approaches rely on observed data from several simulation runs. However, these simulation runs are cost-intensive in time and money. The gathered data is only valid for observed conditions. Using this data to predict the system's behavior under unknown conditions can be erroneous. Furthermore, most automated approaches are restricted to very specific simulation models.

Goals

The concept of this thesis, named *Token Sampling* is an automated approach that overcomes several of the mentioned drawbacks. It is based on the idea to utilize similarities in the processing of tokens in order to reduce the computation time of a simulation run. Instead of processing every token individually, several tokens are handled in the same manner as one reference token. The presented method doesn't need a cost-intensive preprocessing step to gather data. It is an online approach, that analyzes the system behavior at runtime.

At runtime it analyzes the behavior of the simulated system and uses that information to determine system parts which can be replaced. The replacement of these parts is done at runtime. The analysis of the system's behavior is constantly redone to adapt the behavior of the replacement to new conditions. Due to this highly dynamic approach the strength of the effect can be controlled at runtime. Furthermore, the presented concept is capable of adapting to structural changes that may happen at runtime.

Chapter 2

Terminology

This chapter introduces the key terms and their definition. Some of them are well-established and do have different meanings, depending on the context they are used in. Therefore the terms will be defined in the context of this work to provide a base for the discourse following later on.

2.1 Systems and Models

The two terms *system* and *model* are very common, used in almost every scientific domain, and therefore almost everyone intuitively associates a meaning with them [Rop09]. To define the term *model* we first need to discuss the term *system*. After that we will introduce the term *simulation* and some other terms related to this work.

2.1.1 Systems

The foundation for the general system theory was laid by Aristoteles. He already differentiates between the two versions of a *multeity* [Rop09]: Is the order of the components of the multeity irrelevant then it is a set, otherwise an *entireness*, a system. This simple distinction could be found in almost all scientific domains, until in the nineteen-thirties Ludwig von Bertalanffy recognized that these domain specific system descriptions already were very universal descriptions. He then proposed a *general system theory* which was the foundation of the modern, mathematical oriented system theory broadly used today [MT75, Pic75, Rop09].

In this theory a *system* S is defined as a double $(S, \mathcal{V})$ [MT75]:

Mathematical System Definition

$$S \subset \prod_{i \in I} V_i \tag{2.1}$$

Where

- $\prod$ denotes the Cartesian product.
- $\mathcal{V} := \{V_i \,|\, i \in I\}$ is a non-empty set of sets, with I as the (infinite) index set.
- S is a non empty subset of the (infinite) Cartesian product of all sets $V_i \in \mathcal{V}$.

V_i is often called *system object*. It stands for an attribute or characteristic of a system and contains all alternatives in which this attribute can be observed. Therefore, a system S is defined as the set of all observable (and therefore proper) combinations of occurrences of system objects. Because V_i can be a system itself, complex systems can be created in a bottom-up-approach by using simpler systems as *system objects*.

Colloquial System Definition

The literature contains besides the mathematical definition also a lot of colloquial definitions. Those definitions have in common that they define a system S as an arrangement (or combination) of a set of components V_i where the ordering of V_i is characteristic for S [Deu94, VDI96, Rop09, CL06]. Particularly von Bertalanffy noticed that only an intrinsic order of the components of his examined biological systems allowed them to conduct complex processes which exceeded the functionality of the single components.

Additionally to the definition given above, the DIN- and VDI-guidelines require for a system, that it is differentiated from its environment through a set of well-defined criteria. This set is called *system border*. Systems that exchange information across their border with their environment are called *Input-Output-Systems*.

Input-Output-Systems

Mathematically, an *Input-Output-System* is defined by a triple [MT75]

$$(X_\mathcal{V}, Y_\mathcal{V}, S) \tag{2.2}$$

where

- I_x and I_y are subsets of the index set I, i.e. $I_x \subset I$; $I_y \subset I$; $I_x \cap I_y = \emptyset$.
- $X_\mathcal{V}$ is defined as $X_\mathcal{V} := \prod_{i \in I_x} V_i$ and is termed *input object*.
- $Y_\mathcal{V}$ is defined as $Y_\mathcal{V} := \prod_{i \in I_y} V_i$ and is termed *output object*.
- S is a non-empty relation $S \subset X_\mathcal{V} \times Y_\mathcal{V}$

In the following the notation introduced by Pichler will be used. He uses X for $X_{\mathcal{V}}$ and Y for $Y_{\mathcal{V}}$ accordingly [Pic75].

Material Flow Systems

The VDI-guideline 2689 [VDI08] uses the definition of Input-Output-Systems to define *material flow systems* (often also called *production systems*). An Input-Output-System, where at least one of its system objects $V_i \in X \cup Y$ is some kind of a *workpiece* (e.g. substances, pieces, data carriers), is called a *material flow system*. The definition, which system object V_i happens to be a "workpiece", is purely semantic.

2.1.2 Models

Difference between System and Model

Basically, a *model* is a mapping or description of a system, while being a system itself [VDI96, Nie77]. Therefore in system theory a model can be described by Equation 2.1. This recursive definition often results in the synonymous usage of the two terms system and model [CL06]. On the following pages we will try to define the term *model* more clearly and distinguish it from the term *system*. However, throughout the remaining chapters we will use the two terms interchangeably.

In the literature the mapping of a system onto a model is often understood as a simplification of the original system, i.e. that means, the model has only a subset of the aspects of the mapped system [BH97, VDI96, Deu94, LK00]. This understanding is caused, among other things by the fact that real systems often are far too complex to understand, to describe or to be modelled (e.g. we are not able to remodel every aspect of a galaxy cluster system).

Model Validity

Typically a model describes properties and internal processes of the mapped system. The purpose of a model is to be able to study the mapped system under certain circumstances, when applying these to the original system is not possible or feasible. A model is *valid* when the gathered propositions through studying the model can be mapped onto the original system [VDI96, Deu94, LK00]. This aspect will be discussed in detail in the next section.

Material Flow Models

Material flow models are models of material flow systems. As defined before, a material flow system may consist of a set of input-output-systems that are coupled together and process other system objects, called "workpieces". Therefore also a material flow model can be put together in a bottom-up approach by combining simpler models.

Tokens

Normally, a processable "workpiece" in a material flow model is called *token*. Mathematically, a token is a system (2.1), where its system objects V_i are interpreted as attributes. Attributes can be related to physics, like the size or weight, but can also be abstract, like an order number or a processing sequence. Let $(S, \mathcal{V})$ be a system according to equation (2.1) and let $\{V_i \,|\, i \in I_A \subset I\}$ be the set of system objects that are interpreted as token attributes. Then we can define the set $\mathcal{K}$ which contains all proper combinations of attribute appearances and therefore the totality of observable

tokens:

$$\mathcal{K} \subset \prod_{i \in I_A} V_i \tag{2.3}$$

Put differently, every element $k \in \mathcal{K}$ represents a valid (and thus observable) state of a token. Given a production system that creates bicycles in two colors: *red* and *blue*. *Red* bicycles always have odd order numbers while *blue* ones have even numbers. $\mathcal{K}$ is then constructed from two system objects V_c and V_n. $V_c := \{\text{red}, \text{blue}\}$ encodes the color of the bicycle and $V_n := \mathbb{N}$ the order number. Given the information above $\mathcal{K}$ for this specific production system contains the following elements:

$$\mathcal{K} := \{(\text{red}, 1), (\text{red}, 3), (\text{red}, 5), \ldots, (\text{blue}, 2), (\text{blue}, 4), (\text{blue}, 6), \ldots\}$$

$(\text{red}, 2)$ for example is an invalid token representation since red bicycles always have odd order numbers. Therefore, this representation is not contained in the above specification of $\mathcal{K}$. Furthermore, if a bicycles during production reaches different states of assembly, all these states would be contained in $\mathcal{K}$.

It is easy to see that, in order to distinguish tokens, an identification attribute (an order number as in the above example) must be included as a system object. Given such an attribute, $\mathcal{K}$ can be queried for all states a specific token can take. In this thesis, such an identification number is taken for granted. That means, the tokens are objects and while they might have the same attribute values they can be distinguished from each other. For example, due to the order number it is possible to distinguish bicycle *one* from *three* - despite both being *red*.

As stated before, a material flow model performs operations on tokens. Standard operations involve the alternation of attributes, the creation or destruction of tokens and the storage of tokens. Normally these operations need a defined amount of time and are often called *processes* (on tokens).

2.2 The Simulation of Models

Experiments and Simulations

To analyse a system, it is exposed to certain input values representing a situation for which we want to know the systems behaviour. This process is called *experiment* or *simulation* [BH97, VDI96]. As already stated above the analysis of a system is not always feasible (too complex, time consuming or destructive) or not even possible (e.g. the system can be imaginary). In these situations, we design a model by mapping some or all aspects of a system onto it and analyse the model instead of the system.

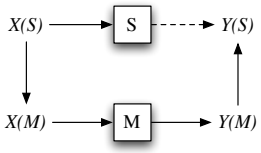
2.2.1 System and Model Composition

More formal, we do have a system S which we want to analyse for a certain set of questions $X(S)$ (also called *system input*). Often the input comprise questions for the behaviour of the model under certain conditions. Applying $X(S)$ to S will then generate a *system output* $Y(S)$ [LK00].

System Input and Output

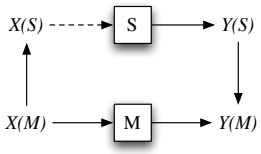
Now there can be three different situations [Pic75] (Dotted lines in the figures refer to things we want to know.):

1. The first one is, that the original system S cannot be used to obtain the output. Then we need to create a model M from the system, transform $X(S)$ onto the model (this is called the *model input* $X(M)$), simulate the model, obtain the output $Y(M)$ and transform it back onto the system, leading to an assumption on the system output $Y(S)$.

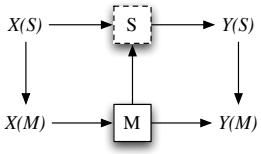


Model Input and Output

2. In the second situation we do have a desired system output $Y(S)$ and we want to know, which input $X(S)$ is needed to obtain the output. This is the search for the cause for the behaviour of the system.



3. In the third situation we do have a model M, an input $X(M)$ and (through a simulation) the output $Y(M)$. We want to create the system S that fulfils the requirements. This is called the *system synthesis*.



Due to the variety of factors (e.g. complexity, system function, questions we want to solve, etc.) there is normally a huge set of models that can be designed as a representation of a system. However, this set of models will be referred to as *model space* $\mathcal{M}$ for a system S. For a more detailed description of the model design process see Law and Kelton [LK00] and Wymore [Wym93].

Model Space

2.2.2 Simulation Types

A simulation can be related to one of three groups, according to the utilised *time set* by the simulated model. A time set (also called clock or clock structure) is specified through a triple $(T, \leqq, t_0)$ [MT75], where

Time Sets

- T is a set, representing points in time.
- $\leqq$ is a partial order on T.
- t_0 is the minimal element in $(T, \leqq)$.

In the following, we will use the symbol $\mathcal{T}$ for the triple $(T, \leqq, t_0)$.

Discrete and Continuous Time Sets

A time set can be either *discrete* or *continuous*, depending on the definition of T. Usually a discrete time set is defined by $(\mathbb{N}_0, \leqq, 0)$ specifying a countable, infinite set of points in time. In a continuous time set, T is specified as $\mathbb{Q}$[1], now making an innumerable, infinite set of points in time available. At those points in time the model can change its state, which is triggered by a *simulation event* executing an *event routine* that is part of the *sequencing structure* of the model [VDI96].

Events, Event Routines and Sequencing Structures

Discrete, Continuous and Hybrid Simulations

Based on the time set used by the simulated model, the simulation is called either *discrete* or *continuous*. If the model uses both types of time set, the simulation is called *hybrid*.

Model State and State Spaces

The *model state* of a model M during a simulation is defined as the state of all system objects $V_i \in M$ at a certain point in time [VDI96, LK00]. The *state space* of a model is defined as the set of all possible states of the model.

A normal discrete, event based simulation consists of four elements:

- The scheduling algorithm. It manages the event queue and executes the events in the correct order.
- The model structure. The structure defines the set of system objects and the connections between them.
- The model parametrisation. It is the initial state of the model.
- A termination condition. If this condition is evaluated as being true, the execution of events is is stopped and therefore the simulation itself.

Material Flow Simulations

Material Flow Simulations are simulations of material flow models. Normally material flow models are based on a discrete time set.

2.3 Complexity Measurement

As stated above a model is often defined as the *simplification* of a system. That means a system S consists of a defined set of *aspects* A_S and the model M features a subset A_M of A_S [BH97, BT00]. In this thesis, however, the model should be *coarsened*. To discuss the difference between model coarsening and model simplification, we first need to define the *complexity* of systems and models respectively.

System and Model Complexity

The term *complexity* can have very different meanings, depending on the considered context. The IEEE[2] [BH97] and Wallace [Wal87] define it in an intuitive way as the degree of difficulty to understand or verify a system. Zeigler [Zei76] specifies the complexity of a system as a value, indicating the amount of resources used to get $Y(S)$. This involves the creation, the

[1] And not $\mathbb{R}$, since it is complicated to represent irrational values in a computer.

[2] The Institute of Electrical and Electronics Engineers

simulation and the analysis of the output. However, we want to accelerate the simulation and therefore we need a complexity measurement in the context of the simulation runtime. Schruben and Yücesan [SY93] argue that the event list management in discrete event based simulations takes up to 40% of the simulation time, leading them to a measurement of runtime complexity based on the minimum and maximum size of the event list. This is a practical definition which may not apply in general or in context of todays modern hard- and software. Zeigler [ZP00] proposes a runtime complexity definition for discrete event models that is based on the quantification of the state space and the *activity* of the measured model. Unfortunate he does not define the term *model activity*.

Runtime Complexity

Because a simple event based simulation algorithm just triggers events, executing event routines, the *activity* of an event based model can be defined as the number of occurring events. When assuming that the execution time of an event routine is asymptotically constant in terms of overall triggered events, an event based runtime complexity for event based models can be defined. While it is possible to over estimate the number of occurring events with the state space of the simulated model, this theoretical estimation is not of interest since most simulation runs are not reflect by it. Complexity measurements for simulations are discussed in Chapter 4.2.2 in detail.

Event based Complexity

Complexity Reduction

Brooks and Tobias [BT00] define the *simplification of a model* as the identification and omission of aspects that are irrelevant for the meaning of the model output $Y(M)$. This means, from an original model M a new model M' without some of the original aspects is constructed, leading to a simpler model. M' is less complex, has a shorter runtime but the simulation results $Y(M')$ do have the same quality as the original results $Y(M)$. Brooks and Tobias and the VDI-guideline note that irrelevant aspects make a model more complex, hard to maintain and hard to analyse.

Simplification of Models

The quality of the model output $Y(M)$ can be colloquially described as the difference in the meaning between $Y(S)$ and $Y(M)$. In other words: The output quality is the difference between the decisions that are made because of the model output $Y(M)$ and because of the system output $Y(S)$ [LK00]. We will refer to the output quality as the error ϵ, introduced by the model relative to the system:

Quality of the Model Output

$$\epsilon = \|Y(S) - Y(M)\| \tag{2.4}$$

The coarsening of a model also removes some aspects of the original model M but this process already starts with a simplified model only containing relevant aspects. Thus the resulting model M'' is a coarsened version of the original with fewer aspects. This leads to errors (a quality loss) in

Coarsening of Models

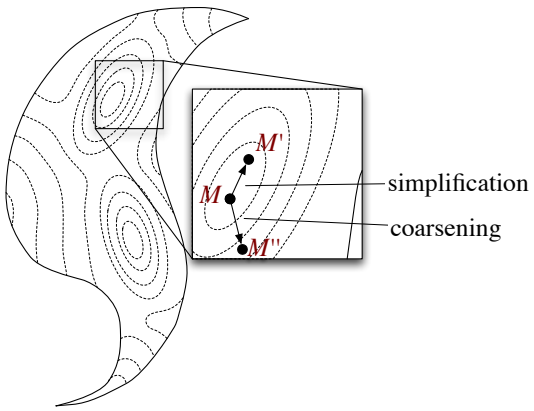


Figure 2.1: The cotyledon proposed by Wymore [Wym93]. It shows the space of all possible models $\mathcal{M}$ representing a defined system S. A contour in the cotyledon represents models with the same quality of model output for a particular input.

the simulation results $Y(M'')$ of M'' when compared to the output of the simulation of M.

The differences between the two methods can be visualised on the *Cotyledons* proposed by Wymore [Wym93]. The following cotyledon (in Figure 2.1) represents the space of all possible models[3] $\mathcal{M}$ for a specific system S and a set of model questions $X(M)$. The space is topologically sorted by the quality of the model results $Y(M)$. A contour represents different models with the same quality of model results.

The simplification process generates a new model M' from an original model M. Because the quality of the results does not change, M' is located on the same contour as M. However, the coarsening of M leads to a new model M'' which is located on a "lower" contour because of the lesser quality of the simulation results $Y(M'')$. The error ϵ in the simulation results (see equation 2.4) can be visualized as the number of contours between the two models M and M'' within the shown space.

Error Metric

The *Error Metric* or *Error Estimation* is the process of predicting the error ϵ for models $M \in \mathcal{M}$ without simulating these particular models. For this purpose the data of the current model M is analysed and quantitative measured [BH97]. A controlling mechanism then chooses, based on the metric, a certain model M' from the model space $\mathcal{M}$ and transforms M to M' by applying a reduction algorithm.

2.4 Controlling a System

Controlling a System

Controlling means to specify a certain (reference) output for a system and

[3] Please note that Wymore uses the term *system design* instead of *model*.

to transform the reference into an appropriate system input $X(S)$ so that the system generates the desired output $Y(S)$.

This leads to the *control law* [CL06]:

$$X(S) = \gamma(R(t), t) \tag{2.5}$$

Where

- S is the controlled system.
- $t \in \mathcal{T}_S$ a point in time.
- $\gamma(.)$ denotes the controlling function.
- $R(t)$ is a reference output.

In literature [CL06, Deu94] two different controlling mechanisms are distinguished: Whether they are *open-* or *closed-looped*. Open-looped means, that the controlling element only receives a reference, leading to the above law 2.5. A closed-looped controlling receives besides the reference output also a *feedback*. Feedback can be understood as a function depending on the system state. This leads to the closed-looped control law:

$$X(S) = \gamma(R(t), u_S, t) \tag{2.6}$$

Where u_S is the current state of the controlled system S.

Chapter 3

Problem Statement

In this thesis a method is presented that accelerates material flow simulations by reducing the model complexity through *coarsening*. It utilizes similarities in the processing of tokens to reduce the computation time of a simulation run. Instead of processing every token individually, several tokens are handled in the same manner as one reference token. This results in a different simulation output (an *error*), but reduces the number of triggered event routines and therefore the overall runtime of a simulation run. An online controlling mechanism is used to keep the output difference small.

The goal is to implement a practical coarsening method. Which means, that first and foremost, it should save resources when applied. Furthermore, it should be compatible with (almost) arbitrary material flow models. There should be no restrictions regarding the design of the model. When applied, the method has to control and adapt itself in such a way that the simulation of the coarsened model still produces valid results in respect to the original model. Here, the original model is defined as the fixed reference to which the coarsened model is compared. Nothing is said about the correct or optimal implementation of the original model regarding the modeled system.

Model Coarsening

Model Coarsening is the omission of aspects from a given model M (cp. Equation 2.1). This creates a new model M' that is said to be less complex and needs less resources for simulation, but also has a different output $Y(M')$ than M when simulated.

The aspects which are to be removed must be well chosen. Otherwise, relevant aspects might be removed, rendering $Y(M')$ invalid. This requires detailed knowledge about the model and its dynamics (behavior). To be practical, the method should gather this information in a cost effective

preprocessing step. In particular the time needed for the step execution counts as computational costs for the whole method. Therefore the preprocessing should be as short as possible, so that the whole method pays off early.

Model Specification

The coarsening method should work with arbitrary material flow models. This includes arbitrary processes within the material flow graph as well as an arbitrary structure of the graph. These requirements have to be considered in the model specifications. Especially the processes should be restricted as little as possible.

Another, more generic requirement is, that the model specification can be implemented into a standard simulator. The colloquial definition of the term *Material Flow Model* given in Chapter 2.1.2 for example is not suitable to be automatically analyzed or executed by a computer algorithm. A good formalized and structured specification must be found.

Controlling the Coarsening Process

The controlling is used to adapt the coarsening method to the model dynamics. This ensures a valid output $Y(M')$. This implies that the coarsening method has to be reconfigured at runtime.

Also the controlling has to have a measurable effect on the quality of the simulation results. The result quality should be higher with controlling than without or when the coarsening is applied randomly.

Validation

A valid coarsened model M' is a model where the simulation output $Y(M')$ results in the same decisions as the output $Y(M)$ of the of the simulated original model M (Chapter 2.3). However, decisions cannot be measured. Furthermore, the difference between decisions cannot be computed. It is very difficult to define what a valid, coarsened model is. For the evaluation of the coarsening method, this term must be defined or at least a measurement for the output quality has to be found.

Evaluation

The method presented in this thesis has to be evaluated for its performance. It has to be implemented in a simulation software utilizing available material flow specifications if possible. For evaluation performance, the software should allow the analysis of different aspects of the method as well as standard ratios like the *lead time* of the material flow system.

In the next chapter the state of technology is reviewed under the requirements stated above.

Chapter 4

State of the Technology

Research conducted so far [BT00, HL99, Hub09, JFM05, VG03, CPB06, VB05, Bar98, Sev90] aims to reduce the structural complexity of a simulation model. The structural complexity typically is a static value directly depending on the model design. However, due to variances in the model parametrization, different simulation runs can utilize elements in different ways, thus changing their importance for the simulation result. Therefore it is essential for these methods to identify elements to process and to correctly estimate their complexity and importance.

In this chapter we will therefore discuss available generic model specifications as well as specifications on which complexity reduction is performed. Having a specification for our simulation model we can start to search for model elements on which we want to perform the reduction. Then we will discuss current solutions to rate the elements regarding their complexity and importance.

4.1 Model and System Specifications

A main aspect of a system description is its specification formalism [ZP00]. While it is possible to describe a system in a natural way as plain text, specifications are much more formalized, like *Systems Theory* [Pic75, MT75] and *Discrete Event System Specification* [Zei76] and *Petri nets* [PR08] (Sinha et.al. [SSL+01] give a compact overview).

4.1.1 Systems Theory

One of the most important developments regarding the description and definition of systems is *Systems Theory*. System Theory is an abstract,

mathematical approach to define and describe systems and their aspects. The system is broken down into mathematical structures like sets and equations. In Chapter 2.1.1 it was already used to introduce the two concepts *system* and *model*. A system is defined as a double $(S, \mathcal{V})$ where S is a subset of the cartesian product of sets $V_i \in V$ (see Equation 2.1). V_i is called *system object* and can be a system itself. Complex systems can then be created in a bottom-up-approach by using simpler systems as *system objects*.

To specify the milling machine from earlier, at first $\mathcal{V}$ must be defined. $\mathcal{V}$ could contain properties like *size*, *material type*, *revolutions per minute* (rpm). Then, a specific milling machine is defined by the combination of the parameter values for each property $V_i \in \mathcal{V}$. For example,

$$m := \{175, steel, 3000\}$$

specifies a milling machine m of size 175 cm with a steel head, rotating with 3000 rpm. System theory uses a top-down approach to define objects and their properties. From a very abstract level the definitions get more and more refined and diversified to complex system descriptions like discrete event systems or linear automata.

Input-Output-Systems

Equation 2.2 introduced input-output-systems. An input-output-system can be visualized as a *black box* that receives some input and produces some output which can be observed. Mathematical functions are simple and abstract input-output-systems. The function $y = f(x) = x + x$ with $x \in \mathbb{N}$ can be specified as an input-output-system $(\{\mathbb{N}\}, \{\mathbb{N}\}, F)$ where F is defined as

$$F \subset \mathbb{N} \times \mathbb{N} := \{(x, y) \mid y = x + x\}$$

The conclusion that a function is an input-output-system may be trivial. However, understanding how fundamental and wide-ranged this concept is (nearly everything is such a system), is crucial.

Time Based Systems

Let's use our milling machine example from earlier. Describing the machine m as an input-output system, it may get some plastics p from which it mills a top cover c. This leads to the following equation describing the transformation process:

$$c = m(p)$$

The description is pretty simple and specifies which output is produced for which input.

A more complex description includes *time* as a factor. With the introduction of time it is possible to describe how long it takes for a milling machine to produce a specific output c. During this time the machine is occupied and cannot process another piece, thus introducing a restriction. Given a production system where several of such machines are coupled together it may influence other machines as they have to wait for the milling machine to finish the work piece. Such an effect is usually termed as *bottleneck* and is of major interest when studying production systems.

Given a time set *TimeSet* the milling machine can be described as

$$c(t) = m(p(t), t) \tag{4.1}$$

where t denotes a point in time. In equation (4.1) not only the transformation function m has been made dependent from time but also the input p and the output c. p and c can be understand as functions for which we observe the values over time, while m defines the relation between them. Pichler [Pic75] refers to p and c as *input-, output-processes*.

To tell, whether the machine is occupied or not, we need to know if in the past there was some plastic inserted into the machine but not yet removed. This is difficult because we are not able to remember anything from the past. This changes with the concept of *states*.

The State Space of a System

We introduced the system concept through a cartesian product on a set of system objects (see Equation 2.1). Some of these objects can be used to remember things like the amount of material we already processed or the point in time we started to work on a new piece of material. Therefore we can define the set of all possible system states U_S for a system $(S, \mathcal{V})$ as

$$U \subset \prod_{i \in I_u} V_i \tag{4.2}$$

with

$$\bigwedge_{u \in U} \bigvee_{s \in S} u \subseteq s \tag{4.3}$$

where $V_i \in \mathcal{V}$. Let S be an input-output-system then the state is a part of the input $I_u \subseteq I_x$ since U affects the output Y. The constraint introduced with Equation 4.3 guarantees that U only contains elements that appear in elements of S. U is called *the state space* of system S.

Usually the system state affects the output of a system. This is implemented by requiring $I_u \subseteq I_x$. The classic definition of the term *input* can be thought of as an impulse from an external source. However, this definition covers everything that is needed to compute the output - even the internal

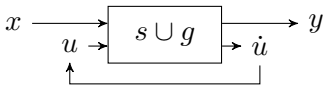


Figure 4.1: Schematic diagram of a general system.

state of the system. Now it is possible to differentiate between the external $\bar{u} \in X \backslash \{U\}$ and internal input $u \in U$. As for the diversification of the input (external and internal) the same concept can be applied to the output. Let $\dot{u}(t)$ be the internal output then $\dot{\bar{u}}(t) \in Y \backslash \{U\}$ is the external output. To ease the syntax we will use x instead of $\bar{u}$ and y instead of $\dot{\bar{u}}$. The general system is depicted in Figure 4.1.

Adding the concept of time, we get a system that is in a particular state $u(t)$ at a certain point of time t. Using equation (4.1) as a basis we get the following equation for a general system:

$$y(t) = s(x(t), u(t), t) \tag{4.4}$$

where

- $x(t)$ and $y(t)$ denote to the input- and output-objects,
- $u(t)$ represents the state of the system and
- $s(.)$ the system itself.

The state of the system may change over time and the changes are normally triggered through the input. This leads to the state equations

$$\dot{u}(t) = g(x(t), u(t), t), \quad u(t_0) = u_0 \tag{4.5}$$

where $g(.)$ is a function that can compute the new state $\dot{u}(t)$ for the system.

With the concept of states it is possible to specify the occupy state u_o for the milling machine:

$$u_o = \begin{cases} \text{occupied} & \text{if} \quad x(t) \neq \emptyset \wedge u(t) = \text{free} \vee \\ & \quad\; u(t) = \text{occupied} \wedge y(t) = \emptyset \\ \text{free} & \quad\; \text{otherwise} \end{cases} \tag{4.6}$$

Interconnections, Subsystems and Components

Until now only whole systems were considered. As stated before, complex systems can be put together from simpler ones. Consider two simple functions $f(x)$ and $g(x)$ then they can be coupled together by

$$y = f(g(x))$$

Figure 4.2: A simple material flow model. Tokens flow from the *Source* to the *Storage*.

This is one of the simplest connections where the output of g is the input to f. More complex systems often differentiate between input (and output) that are available for the connection to other systems and those that are not. In the following some important interconnections between systems will be presented [MT75, Pic75].

Given Equation 2.2, for the input X of a system S the input family can be defined as $\hat{X} := \{V_i \mid i \in I_x\}$. Both, the output $\hat{Y} := \{V_i \mid i \in I_y\}$ and state families $\hat{U} := \{V_i \mid i \in I_u\}$ follow the same scheme. Given a system (S, U, X, Y), then in general the components of $\hat{X}$ and $\hat{Y}$ are available for connection. However, usually most components in $\hat{U}$ are not available for connection. This assumption leads to the class $\bar{\hat{Y}} \subseteq \hat{U}$ that contains all components not available for connection. Using this set definitions $X_{\mathcal{V}}$ and $Y_{\mathcal{V}}$ of a general input-output-system S (2.2) can be specified as

$$X_{\mathcal{V}} := X \times U, \quad Y_{\mathcal{V}} := Y \times \bar{Y}, \text{ where } \bar{Y} = \prod_{V_i \in \bar{\hat{Y}}} V_i.$$

As Mesarovic points out, this allows the definition of a class of connectable systems:

$$\mathcal{S}_{XY} = \{S_i \,|\, S_i \subset (X_i \times U_i) \times (Y_i \times \bar{Y}_i)\} \tag{4.7}$$

Ultimately it is possible to define connection operations within this class of systems. One of the most important connection is the serial or cascade connection $\circ : \mathcal{S}_{XY} \times \mathcal{S}_{XY} \to \mathcal{S}_{XY}$. Let $S_1 \circ S_2 = S_3$ where

- $S_1 \subset X_{\mathcal{V}1} \times (Y_1 \times \bar{Y}_1)$,
- $S_2 \subset (X_2 \times U_2) \times Y_{\mathcal{V}2}$,
- $S_3 \subset (X_{\mathcal{V}1} \times U_2) \times (\bar{Y_1} \times Y_{\mathcal{V}2})$,
- $Y_1 = X_2$ and
- $((x_1, u_2), (\bar{y_1}, y_2) \in S_3 \iff \bigvee_z (x_1, (z, \bar{y_1})) \in S_1 \wedge ((z, u_2), y_2) \in S_2$

Lets explore the class S_{XY} and the $\circ$ connection with a small example of a system (Figure 4.2). This example can be described using the cascade connection:

$$((Source, Conveyor, \circ), Storage, \circ)$$

4.1.2 System Theory of Technology

System Theory is an abstract and formal concept. Ropohl [Rop79] concretizes certain aspects of it and interprets them and relates them to our reality. For example, he categorizes the input and output of a system into three categories: *energy*, *material* and *information*.

Ropohl introduces the term *item system*, which is a loose translation of the german word *Sachsystem*, for systems that can be seen as items produced or intended by humans. Because of this interpretation an item system is embedded into time and space - in the broadest sense. While we introduced time based systems in Chapter 4.1.1 as systems that can be observed over time, Ropohl extends the classic input-output-system definition. Let (X, Y, S) and $(\{V_R, V_T\}, \{V'_R, V'_T\}, S_{RT})$ be input-output-systems where

- V_R and V'_R are system objects that can be interpreted as sets of space coordinates and
- V_T and V'_T are time sets and
- X and Y contain only system objects that are of the classes *energy*, *material* or *information*.

Then, the item system (X_{IS}, Y_{IS}, IS) is defined by the cartesian product of two input-output-systems (2.2):

$$\begin{aligned} X_{IS} &:= X \times \{V_R, V_T\}, \quad Y_{IS} := Y \times \{V'_R, V'_T\} \\ IS &:= \{((x, r, t), (y, r', t')) \mid (x, y) \in S \wedge ((r, t), (r', t')) \in S_{RT}\} \end{aligned} \tag{4.8}$$

with $r \in V_R$, $r' \in V'_R$, $t \in V_T$ and $t' \in V'_T$.

Based on certain constraints Ropohl identifies several classes of item systems. Three of them are of special interest for describing production systems and will therefore be discussed in the following.

Warehousing Systems

Warehousing is characterized by the non-altering of system attributes over time. Let (X_{IS}, Y_{IS}, IS) be an item system (4.8) then it is a warehousing system if and only if

$$\bigwedge_{s \in IS} x = y \wedge r = r' \wedge t \neq t' \tag{4.9}$$

We require, that every input x and the space coordinate r are the exact same as the output y and r' respectively. With $t \neq t'$ we require, that between input and the corresponding output time goes by (though we don't require $t < t'$).

Production Systems

These systems change attributes (or system objects) over time. The milling machine specified earlier may change the material from *raw* to *processed* (Again: These terms are interpretations of some mathematical representations). Therefore, production systems require that

$$\bigwedge_{s \in IS} x \neq y \wedge t \neq t' \tag{4.10}$$

One remark: The definition doesn't say anything about the space coordinates r, r'. This means, a production system *may* change the position between input and output.

Transport Systems

Unlike warehousing systems, transport systems change the space coordinate of a material (or/and of them selfs) while leaving all other attributes untouched. Conveyor and forklifts belong to this class of systems. The constraint is slightly different to equation (4.9):

$$\bigwedge_{s \in IS} x = y \wedge r \neq r' \wedge t \neq t' \tag{4.11}$$

Conclusion

While Ropohl extends the abstract formalism System Theory with close to reality concepts, he does not define a new formalism. Instead he provides definitions of system classes. To define a complete formalism structures (and definitions) are needed that enable the user to describe internal processes in a system. How does the system generate the output? How does the system itself change or change its environment? Such a formalism for discrete event systems will be discussed in the next chapter.

4.1.3 Discrete Event System Specification

The *Discrete Event System Specification* (DEVS) is a specification that was developed by Zeigler [Zei76, ZP00]. The DEVS is a modeling formalism based on System Theory. Unlike Ropohls definitions it provides structures and definitions to specify a whole discrete event system. In DEVS a discrete event system is defined as a set

$$(X, U, Y, \delta_{int}, \delta_{ext}, \lambda, \tau) \tag{4.12}$$

where

- X is the set of input values,

- U is a set of states,
- Y is the set of output values,
- $\delta_{int} : U \to U$ is the *internal transition* function,
- $\delta_{ext} : Q \times X \to U$ is the *external transition* function, where
  - $Q = \{(u, e) \,|\, u \in U, 0 \leq e \leq ta(s)\}$ is the *total state* set,
  - e is the *time elapsed* since the last transition,
- $\lambda : U \to Y$ is the output function and
- $\tau : U \to \mathbb{R}^{+}_{0,\infty}$ is the set of positive real numbers with 0 and ∞.

A DEVS can be in three abstract states and works as follows:

- If an input $x \in X$ arrives at the DEVS, the state $u \in U$ of the DEVS is changed according to $\delta_{ext}(.)$.
- If the time spend in a state e reaches the value of $\tau(u)$ and there is no input then the internal state transition function $\delta_{int}(.)$ is triggered and defines a new state $u' \in U$.
- If neither of the above situations applies the DEVS stays in state u.

With a given input trajectory for a time interval $t_0..t_\#$ the state trajectory $u(t)$ is described as follows:

$$\dot{u}(t) = \begin{cases} u(t) & \text{if } x(t) = \emptyset \wedge e(t) < \tau(u(t)) \\ \delta_{int}(u(t)) & \text{if } x(t) = \emptyset \wedge e(t) = \tau(u(t)) \\ \delta_{ext}(u(t), e(t), x(t)) & \text{otherwise} \end{cases} \tag{4.13}$$

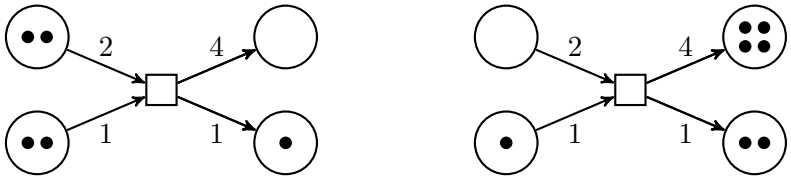
where $e(t)$ describes the elapsed time since the last state transition. For the above equation this means the elapsed time since case two or three applied.

The output function $y(t)$ then can be constructed through

$$y(t) = \begin{cases} \lambda(u(t)) & \text{if } e(t) = \tau(u(t)) \\ \emptyset & \text{otherwise} \end{cases} \tag{4.14}$$

Basically Equation 4.12 refers to a System Theory input-output-system (X, Y, S) (2.2) with states. Its easy to see that the input X and output Y can be adopted. The relation S is then specifyied as

$$S := \{(x, y) \;| x \in X \wedge y \in Y \wedge \bigvee_{\overline{u}} \bigwedge_{i=1,\ldots,n} \begin{array}{l} u_i \in \overline{u}\delta_{int}(u_{i-1}) = u_i \wedge \\ \delta_{ext}(.,.,x) = u_n \wedge \lambda(u_0) = y\} \end{array} \tag{4.15}$$



(a) Initial State. **(b)** State after *firing*.

Figure 4.3: Example of a simple petri net with four places and one transition.

4.1.4 Petri nets

The Petri net is a graphical language to describe parallel processes in systems with many (connected) components. It was invented by Carl Adam Petri to describe chemical processes [PR08]. Basically, a petri net is a directed graph and can be described by a quintuple (P, T, F, W, m_0). The nodes are either from type *place* (P) or type *transitions* (T). The edges of the graph can go either from a place to a transition or from a transition to a place $(F \subseteq (P \times T) \bigcup (T \times P))$. A place can contain an infinity number of tokens. Transitions (as the name says) move tokens from one place to another. The number of tokens taken from a place or added to a place are specified at the edges through a weighting function $W : F \to \mathbb{N}$. $m_0 : P \to \mathbb{N}$ defines the initial number of tokens in each place and is called the initial configuration.

Dynamics

The transitions in a petri net are responsible for changing the state. A transition is said to be *enabled* when each predecessor contains at least as many tokens as the connecting edge specifies. Therefore, the transition in the example in Figure 4.3a is enabled. When a transition is enabled, it can *fire*. In this definition transitions fire immediately. Later on we will get to know more complex firing mechanisms. The firing-process cannot be interrupted. The transition consumes the number of tokens indicated by each edge from the predecessors and adds tokens to the successive places in the number that is specified by the connecting edge. Therefore, in the example, three tokens are consumed, and five tokens are added to the successive places. In the resulting state (cp. Figure 4.3b) it can no longer fire, as in the place in the upper left two tokens are missing.

Timed Petri nets

So far, the petri net definition does not include any clock or time structure. Instead, we said that transitions fire immediately when they are enabled. In *timed petri nets* each transition $v_i \in T$ is associated with a sequence of positive (real) values $t_i = \{t_{i,0}, t_{i,1}, \ldots\}$ with $t_{i,k} \in \mathbb{R}_0^+$. These values are

interpreted as delays for the *firing* of the transitions. The $t_{i,k}$ value defines the delay for k-th *firing* of transition v_i. After being enabled, the transition has to wait the amount of time specified by the delay before it may fire. A transitions v_j reproduces the immediately-firing-behavior by setting all delays to zero: $t_{j,k} = 0$ for $k \in \{0, 1, \ldots\}$.

Conclusion

Petri nets are used in a wide range of problem domains. Based on the initial (most simple) definition of petri nets, formalisms, models and methods have been developed. For example, formal languages, analytical methods (e.g. reachability of states, blocking, etc.), *colored* and *timed* petri nets, just to name a few. Furthermore, they can represent a DES if the transitions are seen as events that are triggered during simulation [CL06]. Therefore, they can serve as a meta-model for any kind of DES model. However, the complexity of petri-net representations tend to explode with the complexity of the models.

4.1.5 Current Simulation Software

Today's simulation software packages offer very powerful solutions for the modeling, simulation and analysis of a wide variety of systems. The software differs in the supported simulation methodology, the modeling paradigms and the usage. At one end, there are highly specialized and optimized software tools, typically used in "Computer Aided Design" or on super computers [Fri03]. On the other end, there exists enterprise simulation software which follows a very general approach. Often the software is split into different products, each covering a specific modeling area, like "logistics", "warehouse" or "airport" [Inc12b] Often coupled with a graphical user interface and a dynamic process animation, the user is allowed to build and analyze simulation models via drag & drop, e.g. [Inc12a, Roc11, XJ 12b]. We will discuss the general approach of most commercial enterprise simulation packages using the examples of *Enterprise Dynamics* build by INCONTROL [Inc12a] and *Anylogic* which is build by XJ-Technologies [XJ 12a]. Modeling a specific system can be done in many different ways. The set of reasonable approaches often depends on the system that will be modeled and the used simulation software. For enterprise related systems, like material flow systems, the *flow-based* modeling has been proven to be easy to use. Flow-based modeling means to connect building blocks through defined inputs and outputs to exchange entities. Each block represents a function that is applied to the entities or stands for a behavior that is triggered by the entities flowing through it. The connected building blocks form a network of functions and behaviors. A simple example of such a network is depicted in Figure 4.4. Here a *Source* creates *Tokens* that flow through the different

Flow-based Modeling

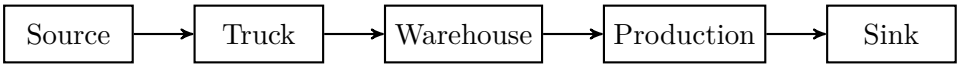


Figure 4.4: A simple material flow model. Tokens flow from the *Source* through the different blocks to the *Warehouse*.

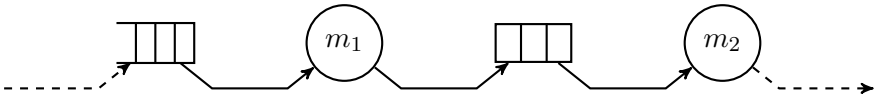


Figure 4.5: The schematic view of a queueing model, representing a manufacturing system with two buffers and two machines m_1, m_2.

blocks, each representing some sort of process, until they reach the *Sink*.

The software product normally consist of a simulation engine which provides abstract implementations of the different simulation formalisms like *Discrete Event Systems*, *Agents* or *Dynamic Systems* [Sof11, XJ 12b]. Because the referenced products all use their own, proprietary simulation engine and modeling mechanisms we cannot state much about them. However, almost all use an object oriented approach.

4.1.6 Representations for Analytical Processing

Until now we got to know model representations that are designed to be simulated, like DEVS. Often it is possible to describe the behavior of an input-output-system through mathematical structures, like differential equations. Then it is possible to derive characteristics of the system solely by mathematical study of the system structures.

Queueing Theory

Queueing Theory is one of the more popular areas. It is a subarea of Probability Theory and Operations Research and covers the mathematical analysis of queues (or waiting lines). Usually a queueing system is represented by a stochastic arrival process and one or more stochastic servicing processes. Other parameters are the maximum capacity of the queue (normally set to infinity), the number of incoming jobs (also normally set to infinity) and the processing order (normally *first come, first served*). These queueing systems are used to model customer lines, traffic systems, telecommunication systems or production systems [Flo95]. Such queueing systems can be combined to form queueing networks. These networks can be used to model whole production systems with several subsystems.

In Figure 4.5 a manufacturing system with two machines m_1 and m_2 is depicted (This example is drawn from [CL06]). Material flows into the system from the left and exits to the right. Between m_1 and m_2 there is a

buffer with a limited capacity of three items. The buffer in front of m_1 is not limited. m_1 with the buffer in front forms a classical queueing system with a virtual arrival process (depicted as the dotted input to the queue), a limitless queue and m_1 as a single server. Two interesting performance parameters are the *waiting time* w_i and the *system time* g_i for the i-th customer (here token), especially their stochastic behavior. w_i is defined from arrival at the buffer until the beginning of the service through m_1. g_i describes the overall time the i-th token stays in the system, i.e. the time from arrival to departure (here to the second buffer). Let $\{w_i\} := (w_0, w_1, \ldots)$ identify the sequence of waiting times and $\{g_i\}$ the sequence of service times respectively. Then the probability distribution of $\{w_i\}$ is defined as $P[w_i \leq t]$. As we can see here, the distribution depends on i. However, usually, for $i \to \infty$ we find, that there exists a stationary distribution, independent from i, i.e.

Steady States

$$\lim_{i \to \infty} P[w_i \leq t] = P[w \leq t]$$

This means, that, if enough token have been processed by the system, every following token has a *stochastically identical* waiting time. This state of the queueing system is often called *steady state*. The same concept applies to $\{g_i\}$.

However, m_1, m_2 and the buffer in between also form a queueing system, with the output of m_1 being the arrival process, the buffer being the queue and m_2 being the single server. Due to the dependency of m_1 of the state of the buffer in front and the virtual arrival process, the analysis of this system gets quite complex. Another aspect that makes the analysis complex is the limited buffer between m_1 and m_2. If m_1 finishes the processing of a piece of material but the buffer is already filled, the machine goes into the *blocking* state. This *blocking* can in certain situations influence to a large part of the material flow. Therefore, even simple systems can be too hard to be analyzable mathematically. Also, analytical methods (like the Queueing Theory) make extensive use of restrictive assumptions on the model. This usually makes the method unusable on more complex models [Hel04, Flo95], like the ones from this thesis.

4.2 Model Simplification and Coarsening

In Chapter 2.3 the terms *model simplification* and *coarsening* were already briefly discussed. In this chapter the discussion will be extended. Afterwards an overview on available simplification methods will be given.

Simplification is an intuitive concept and can be defined as the process of creating a simpler version of a system from a more complex one. In fact, a model M of a given system S is a simplified version of S (otherwise M would just be a copy). Unfortunately there is no single accepted definition [CBP00, BT96] or naming scheme. E.g. some [Sev91, Fra95, SF98, VDI96]

use the term *Abstraction*, while others [SF98, BT00] call it *Level of Detail* and Davis and Hillestad [DH93] use the term *Resolution*. The IEEE [BH97] defines *simplicity* as a "degree to which a system or component [...] is easy to understand". Sevnic [Sev90] differentiates simplification methods into the categories abstraction and *lumping*. Abstraction is used to better understand a system or model. It does not necessarily result in a model with a lowered computational complexity, obtaining simulation results faster. In contrast, *lumping* is used to obtain simulation results faster. Lumped models may be harder to understand than the original model.

Simplification establishes a relation between models [Sev91]. In order to compare two given models M, M' regarding their simplicity two things must be available. At first there must be some sort of validation to determine if M and M' are models for the same system (if they are not, they are not comparable). Second, a metric for the complexity of M and M' is needed to be able to compare them with each other.

4.2.1 Validity of Models

On one hand Chwif et. al. [CBP00] note, that the validation of models receives little attention in the literature and often validation is based on *experience* and *good sense*. As noted in Chapter 2.3 there is a consensus among researchers, that the decisions that are based on the outcome of the simulation of two valid models should be the same [Fra95].

Wymore [Wym93] and Zeigler [Zei76] propose homomorphism as a theoretical basis for the validation of models. Let $M := \{m_1, \ldots, m_n\}, M' := \{m'_1, \ldots, m'_k\}$ be two models, each containing some elements. If a homomorphism h can be found, such that

$$h(m_1, \ldots, m_n) = h(m'_1), \ldots, h(m'_k)$$

then M and M' are models of the same system. The definition requires, that every state, input and output of a system like M, must have a counterpart in the other system, here M'. Unfortunately, this prevents the merging of several states which would require the summation of the time stayed in each state. Sevnic [Sev90] proposes a relaxed definition of validity, the W-validity, based on a homomorphism combined with probabilities. However, the probabilities must be obtained through simulation runs, which make the method computational intensive. Also validity cannot be guaranteed for situations that were not observed during the simulation runs.

Davis and Hillestad [DH93] use the term *Consistency* of models. Two models M, M' are consistent if a set of aggregated states of M can be unambiguously mapped to states in M' (this follows the definition of homomorphism). *Complete Consistency* is achieved when also a disaggregation of the states of M' exists.

Sisti and Farr [SF98] use the term *Accuracy*. They propose two assertions from which they think, they describe how the the terms *Model Complexity*, *Validity* are related:

1. Increased model complexity does not necessarily imply increased validity.

2. Increased computational complexity does not necessarily imply increased validity.

So far concepts were discussed that use the structure of a model to determine the validity of the models. If the validity of two models can be determined this way, the models are valid for all possible inputs, states and outputs. However, homomorphism often is too strict to be of practical use (e.g. it does not allow the aggregation of state) and the user has to create the homomorphic map himself.

Therefore, more practical validation methods compare data, like the model state, retrieved from simulation runs. Because of the state space U of a system S growing exponentially and, like the possible input $X(S)$ of a system being infinitely large, normally only portions of the systems state and the possible input are validated. This is usually done by comparing specific performance indicators during several simulation runs. Do these indicators only differ within a certain threshold the simulation outputs $Y(M)$ and $Y(M')$ are considered to lead to the same conclusions - the models M and M' are both valid and comparable representations.

Sevnic [Sev90] uses the number of packets transported through a network without collision as an indicator for the validity of two models of the same computer network. To analyze material flow systems it is possible to use performance indicators from the field of production controlling. Huber and Dangelmaier [HD09], e.g. use the indicators *Work in Process*, *Lead Time* and *Throughput* to validate three different production models. Johnson et. al. [JFM05] simplify semiconductor manufacturing facilities. To validate the created versions they compare average lead times of products by simulating the original model M and the simplified version M'. If the correlation coefficient for the samples of M and M' is greater than 0.6, they assume that M' is a valid model regarding M. Rose [Ros99] validates the simplified models by comparing the mean, variance and distribution of the lead time of each produced product.

Often the validation of simplified versions of a model is not in the focus of research. Instead, percentages or absolute values are provided to compare simplification methods, by comparing some error metric. The validity of the results of the simple model are not explicitly discussed. Völker and Gmilkowsky [VG03] validate production models by simulating the processing of orders by comparing the mean difference in the start and end times of all orders executed. They use the absolute values to compare different

simplification methods. Hung and Leachman [HL99] present simulation results for simplified models of a wafer fabrication facility. They use the error introduced in the mean lead time of the lots moving through the facility and the standard deviation of the error as indicators to imply validity. Brooks and Tobias [BT00] argue that their created simplifications are valid because they remove or replace components from the model that have no impact on the lead time. They think that, as long as the bottleneck of a material flow model is preserved the results of the models are comparable. Jain et. al. [JLGL99] compare the lead time of high volume products to determine whether the simplified model produces valid results.

Friedman et. al. [FF85] validate the predictive capability of their metamodel (a form of a simplified model) by using coefficient determination and double cross-validation. They split the data they use to train the metamodel randomly into two groups: a training set and a "holdout" sample. Two setups were used: two-thirds and one third and, half and half. The training set is then used to create a metamodel and with the holdup set the predictive capability computed using the coefficient of determination. If this coefficient is high enough, the model is considered to be valid (as long as the assumption holds, that the training set correctly represents the data set the metamodel will be used with).

Fishwick [Fis89] uses different statistical metrics like mean square error or residual sums of error to compare different metamodels regarding their validity. However, he only provides a ranking of the compared methods based on the values of the metrics without interpreting the values regarding the validity of the different methods.

Conclusion

Validation is needed to ensure that the results of a simplified model are meaningful. Usually, if a model is valid, it means, that the results of the simplified model lead to the same decisions as the results of the original model do. Unfortunately, this process is not measurable.

The validation of a specific, simplified version M' of an individual model M is simple. Let $Y(M)$ and $Y(M')$ be the results of simulation runs for the original model, respective a simplified version. If the error $\|Y(M) - Y(M')\|$ is small enough, $Y(M')$ is considered to be valid. Here the validation is limited to measurable values like simulation results and states. The definition of the error metric is undefined and changes from model domain to model domain.

However, there are efforts to mathematically proof that specific simplification methods always retain valid models. That means, that the error ϵ for simplified models never gets greater than a specific value.

4.2.2 Complexity Measurement

Throughout the literature there exists two main definitions of complexity: *Psychological Complexity* and *Computational Complexity*. Psychological complexity can be defined as the degree of difficulty to understand or verify a system [BH97, Wal87, CBP00]. The computational complexity however, usually is defined as the amount of resources needed to get $Y(S)$ from a system. E.g. for simulation model this would be the computational time and memory consumption.

Sisti and Farr [SF98] state that psychological complexity (they use the term *model complexity*) usually is directly related to computational complexity, which again is directly related to computer runtime.

Zeigler [Zei76] specifies the complexity of a model or system as a combination of both psychological and computational complexity. He defines complexity as the resources needed to create, simulate and analyze a model.

Wallace [Wal87] and Huber [HD09] offset write and/or read access on variables against each other to compute a complexity. Brooks and Tobias [BT96] argue that programming code based metrics are not system independent and that the best approach seems to be to identify specific model attributes, like *model size*, *connectedness* or *computational complexity*. Unfortunately, these model attributes are also measured in arbitrary ways. Schruben and Yücesan, for example, use a graph representation of the model to calculate different complexity metrics based on graph theory.

Most literature [Sev90, Ros99, HL99, BT00, VG03] intuitively consider the complexity as the quantity of elements in the model, sometimes combined with the connectedness, represented as the total number of connections between those elements.

Johnson et. al. [JFM05] simply argue that replacing distribution driven machine components by constant delays, reduces the model in its complexity. Chwif et. al. [CPB06] use an absolute value obtained through their model representation and simplification technique to conclude reduced complexity.

Complexity metrics are usually based on the system structure. However, most metrics are not comparable and just offset some properties of the specific model representation utilized.

4.2.3 Simplification and Coarsening Methods

One of the most extensive and most cited taxonomies for simplification methods is provided by Frantz [Fra95] (see Figure 4.6).

He essentially divides simplification techniques into three main categories with each having several sub-categories: *Boundary Modification*, *Behavior Modification* and *Form Modification*. Frantz defines *Boundary Modification* as the modification of the input space $X(S)$ of a system. This includes the explicit removal of certain input objects from $X_{\mathcal{V}}$ whether this is based on

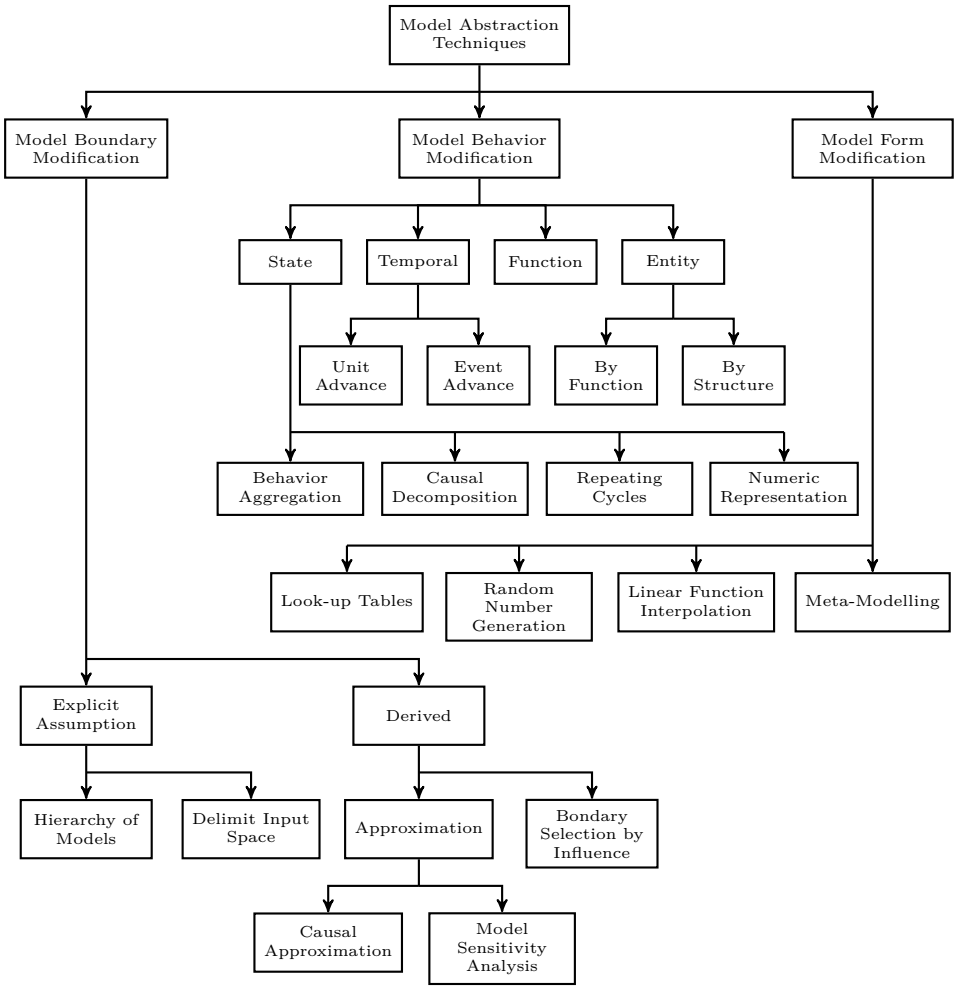


Figure 4.6: Taxonomy of model abstraction techniques [Fra95].

"good sense" or analysis of the model under certain conditions. The second category, *model behavior modification*, involves the aggregation, omission or replacement of states, time, entities or functions. Most simplification and coarsening methods can be located in this category. Entity aggregation is a common method to reduce runtime and/or complexity in material flow models [HD09, Ros07, Ros99, VG03, JFM05, HL99, JLGL99]. Often machines and parts of less interest are replaced by time delaying components. To use *entity aggregation*, one must know how to setup the replacement. In some works, this is done by an experienced modeler. However, especially Sevnic [Sev90] and Huber and Dangelmaier [HD09] each present an automated approach. While Sevnic presents a generic approach based on the DEV specification, Huber and Dangelmaier restrict their approach to material flow models. Usually, information for the replacement setup is gathered in a preprocessing step. During this step the original model is simulated and its behavior can be analyzed and used for the replacement setup. Unfortunately, the results from the simulation only cover the component behavior in observed situations. While this can become a huge problem in unknown situations, it also is a problem when the model itself is changed. Also, this preprocessing step is a normal simulation experiment and needs time and (usually) will produce simulation results we are interested in. Thus, the preprocessing step potentially answers the question we had in the first place, making the simplified model obsolete altogether.

A mathematical simplification technique is the convolution of probability distributions. In material flow systems, process are often modeled using probability distributions as parameters. Several such systems can be combined (or merged) by convoluting the distributions. The convolution is only possible for independent random numbers. Unfortunately, the waiting time of tokens in a material flow system depends on the state of other, connected systems. This dependency leads for example to bottleneck situations. Therefore distribution convolution cannot be used.

The third category is *Form Modification*. This includes e.g. the replacement of the original model or parts of the original model by approximations such as *Meta-Model*s or *Look-Up Tables*. Meta-models are mathematical approximations that take a set of input/output-pairs and try to find a good polynomial fit for them. Then they use the polynomial to compute the output for given input values. The speciality is, that they treat the original model as a black box, completely omitting the internal structure. A good overview over using meta-modeling in production and logistics is given by [Mer05]. Instead of fitting a polynomial to the observed data, Fishwick [Fis89] uses the data to train a neural network.

The method presented in this thesis utilizes *Look-Up Tables*. Basically a look-up table contains a set of specific values. A given input is mapped to an index of the table and the value at that index is used to compute the output. The table may contain output values where the output equals the table

value. Also, the table can contain some intermediate value that is used in a transformation function to compute the output. In the latter case, complex transformations of the input into the output can be omitted. Usually the number of different outputs of the model is far greater than the size of the look-up table. Now the challenge is to fill the table with values that cover the output of the original model as good as possible. Consider the function $f(x) = x \,\%\, 10$ that takes numbers of $x \in \mathcal{N}$ as input and outputs values in the interval $[0, 10[$. It would be a bad decision to replace the function by a table that contains the values of the interval $[20, 25[$.

Examples of models where look-up tables are used are simplified computations of the inverse square-root function $x^{-1/2}$ [Ebe01] or the usage of look-up tables in the color model of the *Graphics Interchange Format* [Com90]. To date, to the author no simplification methods for material flow models utilizing look-up tables are known.

4.2.4 About the Managing of Model States

Until now, most of the referenced methods change the complexity or the level of detail of a given model before it is simulated and not during the simulation. In a pre-processing phase possible changes for a model M are computed which afterwards are applied to create a new, simplified model M'.

Changing the level of detail of a model or a part of a model during the simulation is a much bigger challenge. This is called *multi-resolution modeling*. Usually [HD09, Mue05] several simplified versions of a model or a model part exist. During a simulation run, based on current requirements, the different versions are exchanged. Consider that for the material flow model from Figure 4.4 on page 25 the production building block is available in two different resolutions: c, c'. The original building block c uses physics attributes like friction, weight and size to compute positions of tokens, while the simplified variant c' works like a first-in-first-out buffer with a specific timed delay. Now the model is simulated using c and at a specific point in time t' c is replaced by c'. As c has been part of the simulation so far, c probably changed its state u, i.e. the conveyor current transports some tokens. Since c' has not been simulated, it is still in its initial state, i.e. this component is empty. Simply exchanging c by c' therefore equals a complete reset of the state of this part of the model. The introduced error can be described as the difference between the two states: $\|u_c(t') - u_{c'}(t_0)\|$. Furthermore, when switching back to c at another point in time t'' the component starts where it was replaced at time t'. The introduced error is even bigger. It is not only the difference between the state $u_c(t'')$ and the state $u_c(t')$ where c was left. But its the difference between $u_c(t'')$ and $u_{c'}(t'' - t')$. Because c' cannot fully imitate the behavior of c, c' starts at t' in the wrong state and is only simulated for the time interval $t'' - t'$, the

error accumulates. Therefore it is essential to preserve the consistency of the states of c, c', e.g. by porting the state of c to c' and back when switching.

State Reconstruction

In parallelized simulation, state reconstruction is essential. Parallel simulation means to split the elements of a model into groups, which are then simulated in parallel on different processors (or even computers) [Fuj98]. Because elements of different groups have to communicate and react to each others state change, elements from one group need the correct state of elements of another group. Conservative simulation methods have groups waiting for each other computing the needed state to prevent inconsistencies. Their speed up depends on the *connectedness* of the groups but usually it is relatively small. Optimistic methods on the other side allow inconsistencies. However, they provide mechanisms to detect them and to reconstruct the correct state from a saved state and event history (often called *Time Warp*). Because a state inconsistency can render other (depending) states inconsistent, in the worst case the whole model state has to be reconstructed. A lot of research was concluded to find a efficient state and event history method. Huber [Hub09] notes, that these methods often rely on assumption not applicable for larger simulation models, like small state spaces. Instead, Huber proposes a reconstruction method for material flow models that heavily relies on domain and implementation specific knowledge. He uses knowledge about the structure of the involved components and the scheduled events to compute a state for the component that is switched to. The available information is categories into being related to seizure, failures or statistics. Then for each of the categories and the component types between to switch rules exist how to compute a valid state.

4.2.5 Dynamic Model Simplification

Most of the aforementioned simplification methods have in common, that they create a simplified version of a model that is then being simulated. Because of the domain and result specific nature of the model simplification and validation, simplified models are only valid within bounds. Changing the simulation target or the structure of the original model usually results in a different simplified version of the model. Another problem for the validity of a simplified model can be the model dynamics. Rose [Ros00] notes that the bottlenecks in a material flow model are very important for a valid result. During runtime certain events, e.g. machine failures, can move a bottleneck from one component to another, create new bottlenecks or disperse existing ones. Therefore, it is preferable to adapt the extent of the simplification to the current model state at runtime. In the next chapter methods to detect bottlenecks in material flow systems are discussed.

To the best knowledge of the author only one attempt has been made towards simplification control at runtime. Mueck [Mue05] uses the current position and field-of-view of a virtual avatar to determine which parts of a model should be simplified and to which extent. The simulation models he uses are defined in the three-dimensional space with positions and three-dimensional visualizations for the model components. The user is represented as an avatar in the virtual space. During simulation, model components are simplified based on their distance to the avatar, his field-of-view and whether the objects are visible or occluded by other objects (e.g. a factory wall).

4.3 Bottleneck Detection Methods

As already mentioned the detection of bottlenecks often is crucial to get valid results when using coarsening methods [JFM05]. Results gathered when using the method presented in this thesis also clearly show that coarsening the bottleneck greatly affects the simulation output quality in a negative way (Chapter 8.3). Furthermore, several aggregation methods [BT00, JFM05, Ros00, Ros07, HL99, HD09] use almost the same approach: 1) Identify all bottlenecks in a production system. 2) Aggregate everything else but leave the bottlenecks unchanged. Because of these findings it is necessary to study bottlenecks and the detection of them. A good overview over the field is given by Wang et al. [WZZ05].

4.3.1 What is a Bottleneck?

The literature is quite indifferent about its definition. Lawrence and Buss [LB95] identify three principal definitions found in literature and add their own. The definitions from literature are based on production ratios such as the work-in-process inventory:

- Bottlenecks occur when the demand exceeds capacity in the short run. The detection of these *short-term* bottlenecks is primarily used for process control [LCN09].
- The process with the longest work queue or WIP is the bottleneck.
- Processes with the highest utilization are bottlenecks and are hindering an increased output and throughput in the long run. The identification of these *long-term* bottlenecks is especially used in production planning [LCN09].

Later on, Lawrence and Buss [LB95] state that they define the production bottleneck to be the process with the highest utilization. However, their own definition which they call the economic bottleneck, takes economic ratios into account. In this definition processes are characterized from an

economics point of view (which must not necessarily be the same as from a productions point of view). Kuo et al. [KLM96] provide a fifth definition: They define a bottleneck as a process which's throughput affects the overall system throughput. The largest bottleneck then is the process which has the greatest impact on the system throughput.

4.3.2 Detection Methods

Corresponding to the different bottleneck definitions several detection methods are described. While most methods rely on production values like the machine utilization which are obtained through real world observations or model simulation there are also analytical approaches. Kuo et al. [KLM96] present a method that uses performance assumptions based on probabilities to compare two adjacent machines with each other. Given are two machines in a serial production line. Then, by comparing the possibility of blockage and starvation for each machine, it is possible to specify the direction (up-stream or down-stream) of the bottleneck, in relation to the compared machines. While the probabilities in the studied model can be of theoretical nature, they can also be derived from statistically analyzed (real world or simulation) data. A drawback of this method is the restriction to serial production lines.

Another approach using basically the same information as Kuo et al. is described by Li et al. [LCN09]. However, they focus on *short-term* bottlenecks on real world data. By comparing the percentage of time a machine is blocked with the time it starves (idle state), it is possible to identify the *turning point* (the bottleneck) in a serial production line. The turning point is a machine where the relation between the two ratios (one ratio is larger than the other) turns to the opposite. This approach shares the restriction to serial production lines with the approach by Kuo et al.

Based on personal experience, Sengupta et al. [SDV08] present a method that, according to the authors is less error prone due to its simplicity. The authors write that often performance analysis of real world production systems is difficult because of data errors. These errors can occur because of 1) malfunctioning sensors, 2) incorrect detection of machine states and 3) software problems. Their method analyses the inter-departure times (excluding failure cycles) for a given time period for each machine. Then the *blocked* and *idle* state times are compared between each machine to identify the bottlenecks. However, while their method utilizes a single signal from the machines which is very simple to gather, the method is only applicable on machines with a fixed cycle time.

Brooks and Tobias [BT00] propose an analytical metric to find the bottleneck. Each machine in the simulation model is assigned its *effective capacity*. The machine with the lowest capacity is said to be the bottleneck. The effective capacity is calculated from values like the batch size and the

cycle time and the average breakdown time and the percentage of good parts output by the machine and the repair and setup time. Basically, this single value describes the average rate at which (usable) parts are processed by a machine. Unfortunately, Brooks and Tobias do not examine the bottleneck detection any further, e.g. by comparing the metric to the actual simulation values. Furthermore, the effective capacity does not take model dynamics into account and only works for models that go into a steady state in the long run.

Huber and Dangelmaier [HD09] extend the metric to include a cyclic quotient which puts more weight on component that are included into cycles in the production network. When tested against three different models the metric showed mixed results: For one model the identification was very good while for another one the identification method had the same hit rate as a random process. For the third simulated system the method showed mixed results.

Law and Kelton [LK00] define that the machine with the highest utilization is the bottleneck of a simulated system. While not directly searching for a bottleneck in the simulation model, both Johnson et al. [JFM05] and Hung and Leachman [HL99] also utilize this identification method. Machines with a low utilization are considered to *not be a bottleneck* and therefore can be simplified without the fear of a large deviation from the original model (following the initial findings). However, Wang et al. [WZZ05] note that differences in the utilization of the machines can be very small and that this method is not very accurate.

All the various methods presented so far have some disadvantages in terms of application restrictions, resolution and accuracy [WZZ05, RNT02]. The accuracy was tested by Wang et al. and Roser et al. [WZZ05, RNT03]. For the tested systems the results were identical: A method developed by Roser et al. [RNT02] called *Shifting Bottleneck Detection* was the most accurate one. Furthermore, it overcomes several shortcomings of the other methods.

Shifting Bottleneck Detection

The shifting bottleneck method is able to detect average and momentary bottlenecks. It does not require any knowledge about the network structure and is applicable on more complex production systems [RNT02]. Furthermore, it has been shown to work well with more complex components like *automated guided vehicles* (AGV) [RNT03].

The idea behind the method is, that at any given point in time the machine with the longest uninterrupted *active* period is the most likely the bottleneck. A machine that is active is not blocked (waiting for subsequent machines to finish work) nor does it starve (waiting for precedent machines to finish work). Therefore, the longer a machine is active the more likely

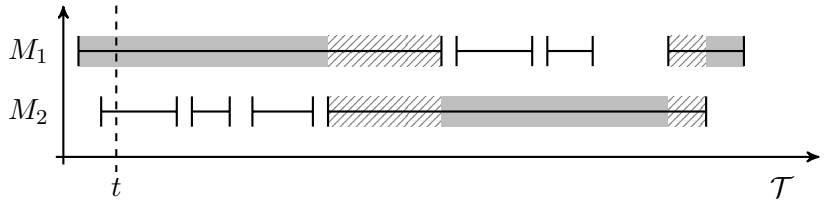


Figure 4.7: Illustration of shifting bottlenecks. Active periods ⊢⊣ for two machines M_1, M_2 are shown. Parts of the periods are categorized where a machine was a sole ═ or a shifting ▨ bottleneck.

it is blocking preceding or starving subsequent machines. However, there are times where a bottleneck shifts (hence the name) to another machine. This is indicated by the overlapping of several time intervals were different machines are stated to be the bottleneck. In such overlapping time periods no machine is the sole bottleneck in the system. Figure 4.7 illustrates the method for two different machines $M1$ and $M2$.

The accuracy of the method was tested by Wang et al. and Roser et al. They used different detection methods to identify the main bottleneck within different systems with varying complexity: Two different production systems with AGVs and a job shop system. Then they tested the sensitivity of the system throughput to changes to the different processes (this follows the bottleneck definition by Kuo et al.). The process with the highest sensitivity was identified as the main bottleneck. It was found that for the tested systems the shifting bottleneck method always found the most sensitive bottleneck. Furthermore, the queue length fluctuates much more than the shifting bottleneck, identifying bottlenecks that last a very short amount of time. On the other hand, the utilization method was handling components with shared resources like AGVs (sharing the track in the tested system) not very well. The method was not able to identify a specific AGV as the bottleneck [RNT03].

While the shifting bottleneck method is the most accurate in terms of the bottleneck definition given by Kuo et al. it also has a drawback: The authors state that for any given point in time t the method is able to identify the sole bottleneck or all shifting bottlenecks. This is true as long as no real time determination is required. For example, in Figure 4.7 at t (without knowing the future) it is not possible to determine whether $M1$ or $M2$ will be a bottleneck. This depends on the length of the active period of the machines in the future. Therefore, this statement is only true if the whole data is known. Otherwise, the classification of the bottleneck component may be changed retroactively.

4.4 Model Partitioning

As we have seen, the usual approach to coarsen a simulation model is through component aggregation. For this approach to work, a valid replacement for a part of a simulation model has to be found (and parameterized). Large parts of a simulation model tend to show a more complex behavior than smaller parts. Due to the connections between the components one component can affect with its behavior the behavior of others (e.g. by being a bottleneck and blocking other machines). Also, large parts may contain active and passive forks that show situation dependent behavior. Formally, the more complex and sophisticated behavior results from a much larger state space. Therefore, it is usually a better approach to aggregate a smaller group of systems than a larger one to minimize the introduced error. Furthermore, sometimes a coarsening approach can only be applied to certain structures within a model. The model must be examined for these structures, e.g. parallel manufacturing lines, and is partitioned into parts that can be coarsened and parts that cannot (those that do not match the set of processable structures). Besides a simpler behavior that has to be examined and reproduced, with several smaller model parts at hand one can locally control the coarsening which may result in a smaller error. Of course, at times rather large models can be reduced to very few components without altering the behavior of the simulation model [BT00]. However, this simplification is done manually by reducing the system state space to represent through extensive domain specific knowledge and analysis of the given system.

Several of the presented coarsening methods partition a given simulation model into several smaller parts. In this chapter different algorithms are examined to partition material flow models and - more generic - graphs.

4.4.1 Partitioning of Graphs

The classic graph partitioning problem is defined as the identification of graph parts that have certain properties. Let there be a graph $G := (V, E)$ with a set of nodes V and a set of edges E and a set of constraints (properties). Then a valid partition is a set of non-overlapping node sets that meet the given constraints. Examples for constraints are: 1) The graph must be partitioned into a specific number of regions and 2) the sets should have about the same size (balanced graph problem). Such partitioning problems (and the problem solving algorithms) are found in very different application areas, such as parallel computing and the detection of cliques in social or biological networks [PN03].

Usually, graph partitioning problems are NP-hard. Therefore, reasonable solutions are based on heuristics and approximation. Even for very special graphs like trees no partially usable approximations exist [Fel13].

As we have seen in the previous chapter some of the bottleneck detection

algorithms focus on serial production lines. Furthermore, as we will show in Chapter 8.5 simplifying serial connected components is less error prone than arbitrary connected components. Therefore, it is crucial to identify such regions in a given graph.

4.4.2 Identifying Sequential Regions

The most simple structures in a material flow graph are groups of sequentially connected material flow components. Probably, these are the easiest ones to simplify. A group of sequentially connected material flow components (L, A, B) (or simply: *serial region*) is specified as

- $\forall S \in L \setminus \{A, B\} : \rightarrow_S \cup_S \rightarrow \subset L \wedge | \rightarrow_S | = |_S \rightarrow | = 1$
- ${}_A \rightarrow \subset L \wedge |_A \rightarrow | = 1$
- $\rightarrow_B \subset L \wedge | \rightarrow_B | = 1$

In most applications, serial regions of maximum size have to be found. Such a maximum sized serial region $\mathcal{R}' := (L, A, B)$ can be characterized as

$$\wedge \text{ serial regions } \mathcal{R}' := (L', A', B') : L' \cap L \neq \emptyset \Rightarrow L' \subset L$$

An algorithm to find maximum sized serial regions is pretty straight forward. Start with a pair of directly connected components $A, B \in G$ in a material flow graph G and a list $L := (A, B)$. Now simply add all neighbors of A and B to L as long as above conditions are met.

4.4.3 Single-Entry-Single-Exit Regions

Besides sequential connected production lines most material flow graphs also contain other, arbitrary structures. Furthermore, some material flow graph may not contain any sequential connected components. Therefore, focusing solely on these structures for simplification restricts the concept to very specific material flow systems.

The concept presented in this thesis can coarsen so called single-entry-single-exit (SESE) regions. Intuitively, a SESE region is a group of components that has a single entry edge and a single exit edge (or node). More formally, a SESE region in a material flow graph G can be characterized by an ordered edge pair (a, b) of distinct edges a and b with the following properties:

1. a dominates b,
2. b postdominates a, and
3. a and b are edge cycle equivalent.

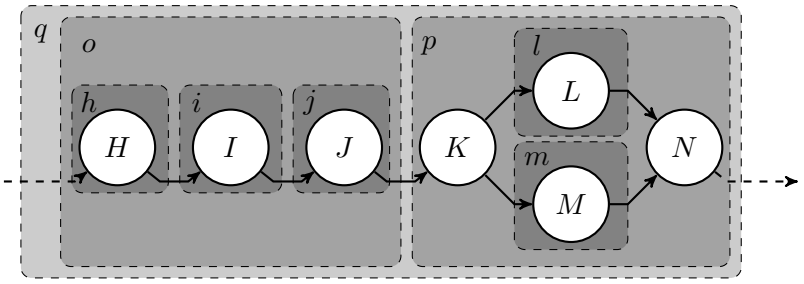


Figure 4.8: Depicted is the part of a graph for which SESE regions have been identified (dashed). The edges that are intersected by the dashed lines are the entry and exit points for the different regions.

In a directed material flow graph, an edge a is said to *dominate* an edge b if every path from a source over b includes a. Similarly, an edge a is said to *postdominate* an edge b if every path from b to a sink includes a. This notion can be easily extended to nodes of the material flow graph in the obvious way.

The first two conditions are necessary but not sufficient to characterize SESE regions. The third condition is required to properly handle loops in the graph, where cycle equivalency is defined as followed: Edges a and b are said to be *edge cycle equivalent* iff every cycle containing a contains b, and vice versa. Similarly, two nodes are said to be *node cycle equivalent* iff every cycle containing one of the nodes also contains the other.

Identifying Single-Entry-Single-Exit Regions

As for serial regions we only want to consider certain regions, i.e. for each edge e in the graph G, the smallest SESE regions for which edge e is an entry or exit edge. In fact, each pair of edges in a serial region encloses a SESE region and a general graph can have $\mathcal{O}(E^2)$ SESE regions. Therefore, we define a *canonical* SESE region (a, b) as

- b dominates b' for any SESE region (a, b'), and
- a postdominates a' for any SESE region (a', b).

With this definition, we can now partition an arbitrary control flow graph G hierarchically into canonical SESE regions since it can be shown that two SESE regions are either node disjoint or nested (see [JPP94]). Figure 4.8 shows an example graph with its canonical SESE regions.

To find all canonical SESE regions of a control flow graph, Johnson et al. [JPP94] showed that the three conditions for SESE regions can be reduced to the single property of cycle equivalence in a slightly modified graph. They showed that in a control flow graph G, edges a and b enclose a SESE region

iff a and b are cycle equivalent in the graph formed from G by adding an edge from *end* to *start*, where a control flow graph is a directed graph with distinguished nodes *start* and *end* such that every node occurs on some path from *start* to *end* and *start* has no predecessors and *end* has no successors. Johnson et al. [JPP94] also showed the very convenient result that cycle equivalence in a strongly connected graph remains the same when removing edge directions.

With this it is possible to find all cycle equivalence classes in a strongly connected graph S by a simple depth-first traversal on the undirected version U of the graph. A depth-first traversal of an undirected graph U will yield a depth-first spanning tree, and the edges of U can be divided into a set of tree edges and a set of backedges. Notice that every circle in U must contain at least one backedge. We call a backedge that connects a descendant of a tree edge t (of a depth-first spanning tree of U) to an ancestor of t a *bracket*. Now, it can be shown that two edges s and t are cycle equivalent in U if and only if they have the same set of brackets in any depth-first spanning tree of U. The set of brackets of each tree edge can be easily computed during an undirected depth-first traversal. When retreating out of a node, we form the union of the bracket sets of the node's children, together with the set of backedges from the node to an ancestor, minus the set of backedges from a descendant to the node. To avoid slow building and comparing of entire sets, Johnson et al. [JPP94] proposed a compact naming scheme for sets of brackets. They basically showed that, when visiting the nodes in a reverse depth-first order and maintaining a stack of brackets, we can characterize the set of brackets of a tree edge by the topmost bracket in the stack and the size of the stack $< topmost\,bracket,\ set\,size >$ (see [JPP94] for detailed information). Listing B.1 shows the algorithm for finding cycle equivalence classes in a strongly connected graph G.

After the computation of the cycle equivalence classes we can identify entry and exit edges of canonical SESE regions during any depth-first traversal of the original graph. The nesting relation between SESE regions can be organized in a tree and we discover them during the same depth-first traversal that determines canonical SESE regions. The depth-first search keeps track of the most recently entered region. When a region is first entered, we set its parent to the current region and then update the current region to the region just entered. When a region is exited, the current region is set to be the exited region's parent, thus forming the tree.

4.4.4 Partitioning of Material Flow Models

Methods that solve the partition problem explicitly for material flow models are very scarce. While not being an explicit partition method Völker and Gmilkowsky [VG03] and Hung and Leachman [HL99] and Johnson et al. [JFM05] use different metrics to separate model components that should

be coarsened and those that should not. Components to be coarsened are identified by a) a low standard deviation of their waiting times or b) they have the shortest processing times or c) they have a low utilitization.

Huber and Dangelmaier [HD09] use a multi-level method with material flow specific conditions to partition the material flow graph. They are able a) to partition the graph into a specific number of groups and b) to keep the complexity of the coarsened material flow within specified bounds and c) to keep the deviation from the original model within specific bounds. For the latter two cases they use a modified version of a complexity metric from Brooks and Tobias [BT00] and a new deviation metric based on the three standard ratios *lead time* and *throughput* and *work in process*.

4.5 The Simulation Software d^{3}fact

The software used to implement and evaluate the concepts presented in this thesis is d^{3}fact [Ren11, REK12, RD13, FRL$^+$10]. d^{3}fact is a discrete event simulation software and collaboration between the research groups *Business Computing, especially CIM* and *Algorithms and Complexity*. It is designed from the ground up to be extendable and its architecture supports user-collaboration. The former is achieved through the usage of a service-oriented architecture, the latter through a client-server approach. d^{3}fact consists of three major programs connected via network (cp. Figure 4.9). The simulation platform contains the simulation kernel and actually runs simulations of models and whole simulation experiments. The visualization client is a program that can visualize data from the simulation platform forming an interface between the software and a human user. The server is a central program connecting the different clients and platforms with each other and also provides access to different resources like 3D data, simulation models and server-side files. This program architecture especially allows the execution of the simulations in a different location than the visualization client. For example, the models might be simulated on high-end server hardware located at the business headquarters while field workers can use the visualization client to show simulation results to customers.

4.5.1 The Server

The server is the central hub. Clients and platforms use it to find each other and to access resources. It is build on top of the OSGi platform [MVA10] in a *service oriented* approach. The OSGI platform is most notably known as the basis for the *Eclipse Integrated Development Environment* and as a framework for automobile systems. This makes the server extremely stable and extendable with new program features. For example, it is very easy to implement support for custom databases or enterprise resource management systems as a source for additional resources. The server integrates itself as a

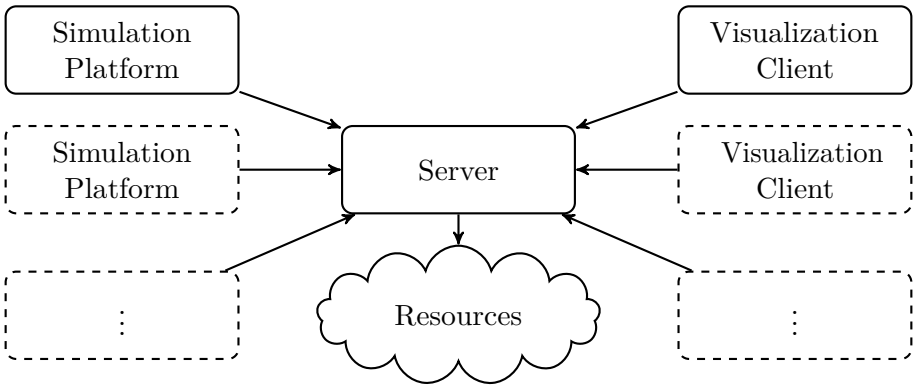


Figure 4.9: The architecture of the d^{3}fact platform.

simple service accepting incoming connections. Connections are represented as services throughout the platform which has the advantage that broken or stale connections do not affect the overall stability of the server.

As depicted in Figure 4.9 is the server capable of handling several simulation platforms and visualization clients at once. This architecture allows the server to run several simulations in different simulation platforms at once. This is especially useful when executing a simulation experiment with a large number of individual simulation runs. The server can distribute the runs over a set of computers (e.g. a high performance computer cluster), speeding up the overall experiment execution. Also, several visualization clients may connect to the same simulation platform to collaboratively edit the simulation model currently loaded.

4.5.2 The Simulation Platform

The simulation platform contains the simulation kernel and executes the simulation models. As the server, it is based on a flexible and extendable application platform. Libraries with additional simulation components can be loaded at start up as needed. Some of the major libraries are:

- Motionplanning
- Physics Engine
- Railcab [HRS$^+$07] Support
- Experiment Design
- Statistical Analysis
- Network Support

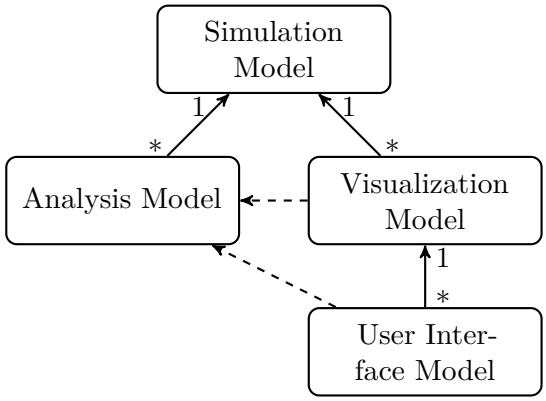


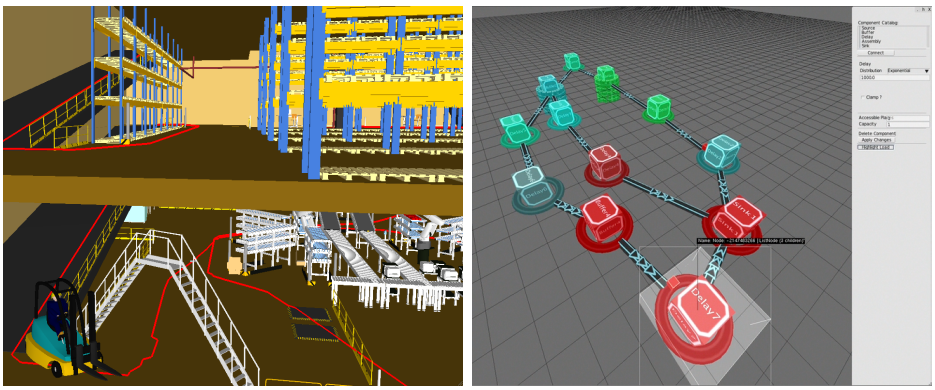
Figure 4.10: The relationship of the implementable models currently supported by d³fact.

First the model architecture is presented as its structure is heavily utilized when implementing the concepts of this thesis (cp. Chapter 7). After that, both the experiment and the statistical analysis will be discussed in detail. These libraries are used to obtain the simulation results presented in the validation chapter (Chapter 8).

4.5.3 d³fact Model Architecture

One major feature of the simulation framework is the separation of the simulation model and its analysis and visualization (cp. Figure 4.10) [REK12]. This two-staged architecture allows the usage of several visualizations with one simulation model making it easy to customize the view for specific parts of the simulation model. Also this enables the execution of the simulation without any visualization and to provide additional model extensions like the statistical analysis without visual clutter.

As shown in Figure 4.10 a simulation model can have several analysis and visualization models. The analysis model describes which values of the simulation model are of interest and provides a standardized way to access them. This allows the analysis of arbitrary simulation models and even the transformation of values into different units when needed. The analysis model is presented in detail in the following. The visualization model describes how the objects of the simulation model should be presented to the user. Since the simulation model can be arbitrarily complex, a formalized approach is needed to transform complex operations and calculations of the simulation model into user-friendly visualization objects (and updates) such as geometric shapes and transformations. Figure 4.11 shows two different visualization models for the same simulation model. While one represents simulation objects as complex CAD models, the other one uses simple geometric shapes to highlight the material flow graph.



(a) Visualization Model representing simulation objects with CAD models.

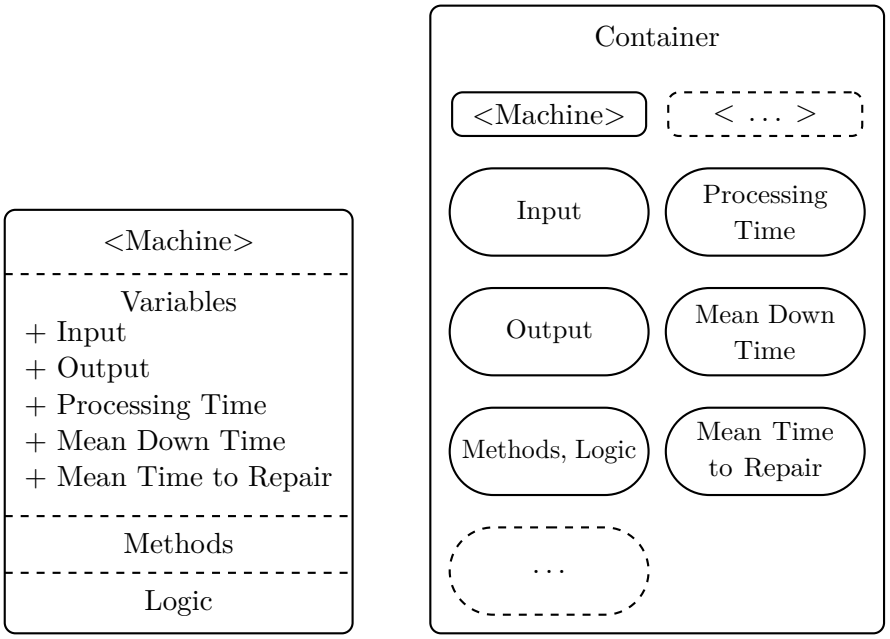
(b) An abstract view on the material flow with the machine load encoded in the component coloring.

Figure 4.11: Two different views of the same simulation model.

Based on this model a user interface can be specified that describes how the user can interact with the visualization objects. The user interface may, for example describe an editor that is specifically designed for the simulation model domain to be very user-friendly. Furthermore, for different users different interfaces (and visualization models) can be loaded to implement usage and access restrictions. Both, the visualization and user interface model can access an analytics model to provide advanced visualizations of refined statistical data.

The Simulation Model

The simulation model specification in d^{3}fact is based around properties and a concept called *composition and aggregation*. This removes the need for a static type hierarchy in the component definitions. Static inheritance type hierarchies are simple to understand as it is natural for us humans to arrange objects in a taxonomy [Som04, SG96]. Unfortunately, they can become hard to maintain because of the limited possibilities for enhancement. Assume we want to categorize *animals* to describe their feeding habits briefly. The two obvious categories are *herbivore* and *carnivore*. Additionally, there are animals that are *omnivores* which derive their habits from both, the herbivores and carnivores. Implementing this hierarchy is tricky in most modern programming languages as most type hierarchies are required to be clean trees. The alternative would be to create a new type omnivores which is not derived from the other two categories. This, however, is a simple workaround because obviously omnivores are also herbivores and carnivores (or at least derive their habits from them).



(a) Standard Object Type. **(b)** Container Concept.

Figure 4.12: This Figure shows the differences between a normal, static type (left) and our container concept (right). Object types are indicated by rectangles and the properties by rounded boxes.

A solution to this problem is to abandon static type hierarchies altogether. This can be achieved with the *composition and aggregation* concept [Gre09, Dea05]. This concept is based around objects that simply have *properties*. Therefore, it is possible to assign to an object very different properties and *several* types, which solves the problem for the *omnivore* type. Instead of derivation, an animal of type *omnivore* is packed with properties from both, the *herbivore* and the *carnivore* category and is additionally assigned with these two types. Figure 4.12 shows the standard static type implementation of a machine material flow component and the same component composed with the *container and properties* approach. The static typed object is of the type *Machine*, has several variables and some publicly available methods. The logic of the object processes events and updates the state of the machine accordingly. The container based object however also has all these properties, methods and logic but, as indicated by the dashed shapes can be updated dynamically with additional properties and types. That makes this approach much more flexible and allows the combination of different types (and properties) which can be near impossible with the static type approach.

In d[3]fact an object is represented by a *container* assigned with a dynamic set of *properties*. The container type provides methods to manage its proper-

ties (add, delete, get by key, etc.). Simple properties like numerical values or strings are passive, meaning they do not react to state changes and also do not cause them. These properties are *aggregated* (weak ownership): The container object owns them but they are not bound to the life cycle of the container object.

The object-specific logic on the other side is an active property, because it does react to state changes. For example, when an event is caught, the logic processes the event causing state changes within the object. Such properties are *composited*. Composition strongly binds a property to a container, which means the property is bound to the container's life cycle. That means, the property is created together with the container and is also destroyed The property can access it's parent (the container) and the whole simulation model, as well as process simulation events.

Simulation objects may have other simulation objects as children. This makes it easy to construct hierarchies within the simulation model.

The Analysis Model

An arbitrary simulation can be very complex in its structure and data representation. An analysis model is the approach to organize and access such data in a standardized way. This allows the automated processing and (foremost) statistical analysis of such data. *Processing* can be the visualization with graphs or diagrams or the storage in an organized way for further processing by external programs.

A date (or value) that changes over the simulated time is called a *series*. Examples for series are the current warehouse stock or the velocity of a simulated car. The model also supports the implementation of complex value calculations, e.g. the average throughput over a fixed period of time. The idea is of implementing such calculations into the analysis model is that, when the analysis is not loaded the calculations are not needed and are therefore not loaded and executed. Series can be organized in simple nodes. Such a node can be thought of as a data sheet where each column represents a series and on each value change a new row is added to the sheet.

The nodes containing series can be further grouped together and be organized in a tree hierarchy. Now a node represents a set of series that belong together, e.g. because the series belong to the same simulation object. This information can be used to group the sets of series by colors.

The current implementation provides support for two-dimensional charts (several types of charts are available) which provide live data updates from a simulation. Writing the data into files is also supported. The actual write process uses application platform mechanisms so that a remotely started simulation platform transfers the data to the server where it is written into files. This especially comes in handy when utilizing computer clusters to execute experiments.

4.5.4 Material Flow Specification

d^{3}fact comes with a component library with which material flow systems can constructed. The current specification follows the same *black box* approach as other simulation software (cp. Chapter 4.1.5). Components do not follow any specification, instead connections between such components are specified.

A connection consists of two parts: The *output* and the *input port*. Any two such ports can be connected and a material flow component may have an arbitrary number of them. While an input port is connected to one output port at most, an output port may be connected to an arbitrary number of input ports. To follow token movement the ports broadcast an event when a token moves through them. However, due to the black box approach, component state changes and token movement (or reordering) within components cannot be tracked in general.

4.5.5 Experiment Design in d^{3}fact

d^{3}fact simulation experiments are specifications of simulation models and their initialization parameters (configurations) and how to evaluate them. For statistically valid and meaningful simulation results it supports the iterative execution of a specific model configuration with different initial random number generator seeds. The framework supports adaptive optimization techniques to speed up the examination of a large set of different model configurations (configuration space). To support the early pruning of the configuration space, the simulation of a specific configuration can be analyzed at runtime. If the results turn out to match a specific termination criteria the simulation can be stopped early. This allows for example the termination of simulations which have arrived in a steady state. The framework supports the generation of new configurations based on previous simulation results. This allows the implementation of adaptive optimization algorithms. Of course, the d^{3}fact experiment specification does not only support dynamic optimization (online analysis, early termination, etc.), but also allows the (automated) execution of completely predefined experiments.

A d^{3}fact experiment contains several *experiment units* which then contain several *simulation scenarios*. A scenario consists of a simulation model and an initial configuration. Each scenario is executed a specific amount of times. This is done to retrieve statistically valid simulation results. An experiment unit is used to group different scenarios together and compare them online. To support early pruning, scenarios as well as whole experiment units can be terminated at any time. Furthermore, to support the generation of configuration from previously optioned simulation results, new experiment units can be added for execution at any time.

Because of the extensibility of d^{3}fact, it was easy to integrate the experiment design framework as an additional plugin. The implementation

supports two different operating modes:

Local All experiments are started within one simulation platform process. Usually in several parallel running threads. Results are analyzed locally but stored at the server.

Global The experiments are distributed across several simulation platforms. Each simulation platform can again execute several experiments in parallel using threads. Results are streamed to a global controller (residing in the server), which analyses and manages the experiments. While this implementation is more complex it has the power to utilize large computer clusters.

4.5.6 The Visualization Client

Currently one client is available: A powerful C++ Client which can handle huge CAD model based 3D scenes [EJP11]. It recently gained low end hardware (netbooks and smartphones) support [EJF10] which is support through the implementation of low-end visualization models. It is developed at the research group *Algorithms and Complexity*. The client is able to display simulation objects as 3D meshes and furthermore can also provide the user with a graphical interface. The two screenshots from Figure 4.11 show the client in action.

A second, web-based client is currently under development. It will also show 3D data and provide a user interface and is based entirely on the web standard HTML5. This client allows the display and manipulation of simulations directly within a web browser.

Chapter 5

Required Actions

The goal is to develop and implement a practical coarsening method. Which means, that first and foremost, it should save resources when applied. Furthermore, it should be compatible with (almost) arbitrary material flow models. There should be no restrictions regarding the design of the model. When applied, the method has to control and adapt itself in such a way that the simulation of the coarsened model still produces valid results.

In Chapter 3 requirements to meet and problems to solve were discussed. Chapter 4 presented solutions that are already available. Now these solutions will be discussed regarding their applicability.

Model Coarsening

Model Coarsening is the omission of aspects from a given model M. This creates a new model M' that is less complex and needs less resources for simulation. The aspects that should be removed must be well chosen. Otherwise, relevant aspects might be removed, rendering $Y(M')$ invalid. Here, the original model (in lack of other references) is taken as a fixed reference to which the coarsened model version is compared. There are no statements made regarding the validity of the original model in reference to the modeled system.

Most of the model coarsening solutions presented in Chapter 4.2.3 either use a manually created simpler version of a model, or need some initial knowledge about the model provided by an engineer. There are only a few works on automated simplification. All require that the original model and often also the coarsened model variants are fully simulated in a preprocessing step. Other applicable modeling approaches like neuronal network metamodels also need this preprocessing step for training. During this preprocessing step,

data on the system's behavior is gathered, which is then used to parameterize the coarsening method. Since this step involves the simulation of the original model, it takes a considerable amount of time. Therefore, these methods are only practical when using the same simulation model for a long time or for a lot of experiments. Furthermore, when in use, the methods are bound to the data from the preprocessing step. Data updates are not possible.

A new method must be developed which does not rely on data gathered from the simulation of the original model. Instead, this method should analyze the model during runtime and instantly utilize that data. This implies, that the method cannot rely on large amounts of data. Furthermore, a lengthly preprocessing step must be avoided.

Model Specification

The coarsening method should work with arbitrary material flow models. This includes arbitrary processes within the material flow graph as well as an arbitrary structure of the graph.

Material flow specifications found in current simulation software typically uses a *black box* approach. This ensures that arbitrary material flow processes can be implemented. A drawback of this specification is the missing formalization of the processes. Without a formalization it is difficult to process (and analyze) such specifications automatically. This also explains why most material flow coarsening methods are restricted to very specific material flow processes.

Another solution would be to rely on discrete event system formalizations like DEVS or (in a more general view) timed petri nets. Since these formalizations describe processes on an event-based level, arbitrary DES processes can be fully specified. However, these specifications are far too general. Material flow specific assumptions or information is not available on this modeling level.

Instead, in this thesis a formalized specification for arbitrary material flow processes will be developed. This specification allows an efficient and automated processing of the whole material flow model (through the utilization of certain material flow specific properties). Besides, this specification can be transformed into the much more general DEVS and petri net formalism. However, this is not necessary for the coarsened method to work and is therefore omitted.

Controlling the Coarsening Process

The controlling is used to adapt the coarsening method to the model dynamics. This ensures a valid simulation output $Y(M')$. This implies that the coarsening method is reconfigurable at runtime.

Most methods try to control the creation of the coarsened model during

the aforementioned preprocessing step. Coarsened model versions simulated to check whether the results are valid within predefined bounds. If this is not the case, the coarsened version is discarded [HD09, CPB06]. Since the coarsening method presented in this thesis does not create coarsened versions of a model in a preprocessing step, controlling must be done at runtime. To the best knowledge of the author, only one attempt has been made towards coarsening control at runtime. Mueck [Mue05] uses the current position and field-of-view of a virtual avatar within a three-dimensional representation of the simulation model to determine which parts of the model should be coarsened. Mueck's method heavily relies on three-dimensional data which is not always available.

As stated before, the coarsening method presented in this thesis has to work with very sparse data. Furthermore, the controlling is responsible to adapt the method to the current conditions (within the model). In models with fast or often changing conditions, the coarsening model must be adapted on a fine granular level, this is especially due to the sparse data. Using the typical approach by simply using the average over a long period of time to approximate a certain condition is not possible. To ensure a fine granular controllable method, two essential parameters will be available: A parameter to control the length of time the coarsening method is applied and a parameter that specifies where the coarsening should be applied. The first parameter essential is a value of the underlying time set $\mathcal{T}$. This guarantees a very fine granular control. However, simply controlling the length of the application of the method is often not sufficient. Consider a simulated material flow system where different parts show very different behavior. Obviously, the controlling process should also take this into account when adapting the coarsening method to the current conditions within the model.

Model Partitioning

This can be done by partitioning the material flow graph into different regions. Huber uses a multilevel approach that solves the general graph partitioning problem on a model complexity approach. This approach partitions the graph into several regions of similar complexity [HD09]. Others [JFM05, VG03] use performance ratios like the machine utilization during analyzed simulation runs for partition.

The method presented in this thesis is restricted to single entry single exit (SESE) regions. Therefore, the *program structure tree* algorithm from Johnson et al. [JPP94] is used to find all SESE regions within the material flow graph. The algorithm is fast, considering that the general graph partitioning problem is NP-hard. The algorithm is capable of finding all regions within linear time. Furthermore, it creates a hierarchy of regions. This is especially interesting as it allows the controlling process to choose between different combinations of coarsened regions.

Validation

In this thesis, the given original model M is taken as the fixed reference against which the coarsened variant M' is measured. Again: No statement is made about the correctness or validity of the original model in reference to the system it models. Therefore, a valid model variant M' is one for which the simulation results lead to the same decisions as for the simulation of the given original model M. Unfortunately, decisions are hardly measurable or comparable. For the concept evaluation, a different measurement must be found. Throughout literature (cp. Chapter 4.2.1) differences in standard ratios like the *lead time* or *throughput* of tokens or the overall *work in process* are used for validation. These ratios are gathered throughout a simulation run for both, the coarsened and the original model. The comparison is usually done in percentages. Like Huber [HD09] all three mentioned ratios are used for evaluation. However, instead of mixing the ratios into one single value, the three ratios will be compared independently. This will show if the presented method affects the ratios in different ways.

Evaluation

For implementation and performance evaluation, the simulation software d^{3}fact will be used. Because the software is developed at the research group *Business Computing, especially CIM* the software and its inner workings are well known. Furthermore, the software can easily be extended and modified where needed, due to its flexible *service oriented* architecture. Also, the software has an integrated experiment and analysis framework that can be used for automated performance evaluation. However, the material flow system specification currently used in d^{3}fact is not usable, as it lacks process formalization. Therefore, a new specification is designed in this thesis. The new specification will be designed as a replacement for the current specification. That ensures, that it will not be solely designed for the needs of the presented coarsening method, rendering the specification useless for other projects.

Conclusion

Current coarsening methods lack the ability to react to model dynamics as they are designed to use static, precomputed coarsened model versions (or model parts). Furthermore, model specifications currently in use are missing a formalization of material flow processes which makes it hard (to impossible) to support arbitrary material flow processes. A controlling based on the model behavior is unknown.

The method presented in the next chapter overcomes several of the mentioned drawbacks. First of all, it doesn't need a cost-intensive preprocessing

step to gather data. It is an online approach and analyzes the system behavior at runtime. The coarsening is applied to regions of the material flow graph at runtime. The analysis of the system's behavior is constantly redone to adapt the coarsening to the current conditions.

Chapter 6

Conceptual Design

In this chapter a concept is presented that accelerates material flow simulations by reducing the model complexity through a coarsening process. The concept is based on the idea to utilize similarities in the processing of tokens to reduce the computation time of a simulation run. Instead of processing every token individually, they are processed in the same way as some reference tokens. This introduces an error in the material flow of the model but reduces the number of triggered event routines and therefore the overall runtime of the simulation. Different to current automated techniques, this method does not need a time intensive preprocessing step. Furthermore, the method can adapt itself to state changes and structural changes made to the simulated system at runtime. An online controlling mechanism is used to keep the error at a minimum while maintaining a valid model output in reference to the original simulation model.

6.1 Token Sampling

As described in Chapter 4.2.3, a discrete event simulation can be speedup by omitting the creation (and processing) of events. With this in mind, the state definition (6.8) and the event set from Equation 6.14 imply three possible areas where runtime can be spared:

1. Computations that are triggered by events can be reduced. For example, a probability distribution can be replaced by a fixed value like its expected value. Unfortunately, for such common calculations like distributions the savings effect is negligible as most of the runtime is consumed by other state changes. Only in very specific situations it is possible to save runtime using this approach. This approach collides

with the requirement of generality of the whole concept and will not be explored any further.

2. State changes that have no impact on the simulation output may be reduced to a minimum. For example, a warehouse may occasionally trigger internal housekeeping processes. Unfortunately, the simulated system ignores the position of the stored goods when accessing them. Then the original warehouse may be replaced by a simple buffer that reduces or omits these computations. Again, these are very specific situations where such an approach pays off. Furthermore, these situations are hard to detect with algorithms, as this requires very specific knowledge about the simulation model.

3. The movement of tokens between components can be reduced. For example, we may replace a group of token processing systems with a well parametrized single system. This allows us to omit any movement within the components and also between components. Brooks and Tobias [BT00] reduce a model of fourteen components to a bare minimum of three, while preserving the original simulation output.

The approach presented in bullet point three is the most commonly used one for model coarsening (see Chapter 4.2.3). It reduces the number of components for which state changes need to be computed (Bullet Point 1), the number of connections, reducing internal state changes (Bullet Point 2) and the movement of tokens (Bullet Point 3). Token Sampling picks up the idea of replacing a part of the material flow by a single component. But instead of costly computing the parametrization for the replacement prior to the actual simulation, this approach will take samples from the material flow at runtime and use these samples as a parametrization hint. This approach has several advantages:

1. This approach can be used while running the original simulation model.

2. It even can be added during a simulation run.

3. Since the sampling does not rely on a static material flow, it can adapt itself to changes made during the simulation run.

4. It does not need a time-consuming preprocessing step.

The following problems need to be solved:

1. Every now and then the components have to be re-sampled. That means a fast approach for switching between the original and the coarsened components must be implemented.

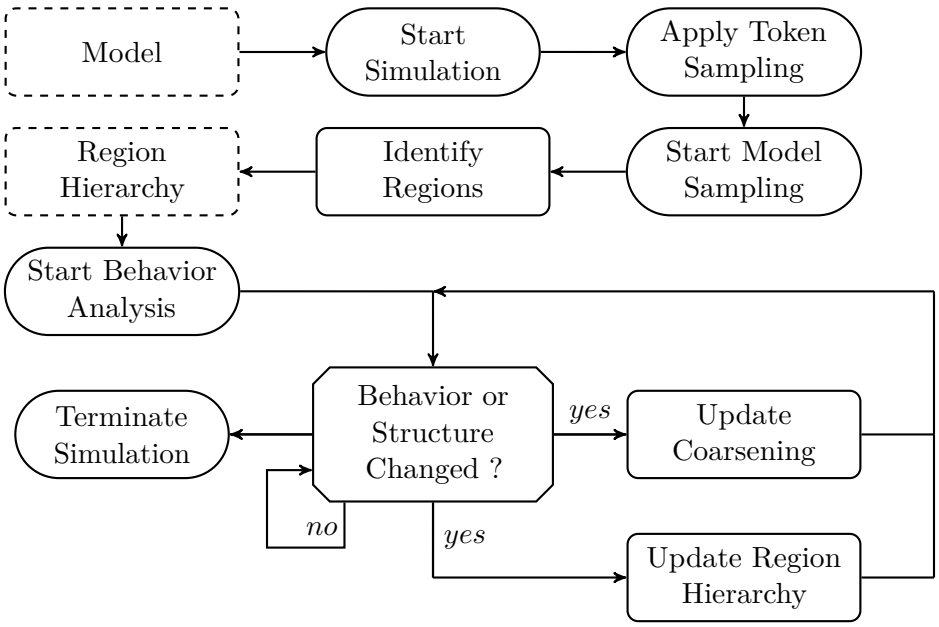


Figure 6.1: Workflow of the Token Sampling Concept. Dashed nodes depict data, rectangles illustrate processes and rounded rectangles define the start or stop of systems.

2. Due to the dynamics of the model, changes that render the current samples invalid, have to be identified during simulation. Then the model must be re-sampled under the changed conditions.

3. From the samples, a good replacement for the original part of the model must be computed. This is needed to gain good quality simulation results.

4. The overall control of the coarsening process should take the dynamics of the model into account. This is also needed to obtain good quality results by maximized speed gain (efficiency).

Concept Integration

In the general, the concept works as follows (also see Figure 6.1): Given is a simulation *model* of a material flow system. The coarsening concept is designed to work with arbitrary material flows as well as arbitrary material flow components. Especially to achieve the latter, a simple, yet powerful material flow model specification has to be used. Such a specification will be presented in the next chapter (6.2).

The simulation of the system may be started in advance, since this concept can be applied at runtime. Once applicated it performs a simple

setup routine. Next a process is started which takes (and stores) samples from each material flow component during runtime (see Chapter 6.3). These samples are needed for a proper state reconstruction when switching between the original and the coarsened model part.

When the component sampling has been setup, the current structure of the model is analyzed for single-entry-single-exit (SESE) regions (*Identify Regions*). The concept is then able to coarsen arbitrary SESE regions. For identification, a modified version of the *Program Structure Tree* (PST) algorithm by Johnson et al. [JPP94] is used. The result is a tree that contains all of the found regions and furthermore, exposes their hierarchical order (*Region Hierarchy*). In general, two types of regions are differentiated: The ones where material flow components are connected in a sequence and every other region. The modified PST algorithm is presented in Chapter 6.4.

For each of the regions a *Behavior Analysis* is set up. This analysis provides metric values that are used to control the coarsening process. For example, the analysis contains a bottleneck detection to prevent the coarsening of regions with bottlenecks, as this usually introduces large deviations in the model behavior. With the analysis in place a controlling algorithm (see Chapter 6.7) can decide which of the different regions it should coarsen. Chapter 6.5 and 6.6 describe for the two region types how the coarsening is applied. Switching between the original and coarsened model components allows the concept to react to the dynamics of the model which includes the handling of unknown situations.

6.2 Material Flow System Specification

Typically, a material flow model consists of token processing systems that are arranged in a graph (cp. Chapter 2.1.2). Simple objects, usually called *tokens*, move through this network from one component to another. While a token stays at a component, the component can perform different tasks on it, e.g. alter its properties. This colloquial description needs to be formalized as only this will provide a strict, complete and verifiable description of such a model and its dynamics [BH97]. To be able to completely and accurately describe the concepts and algorithms in the next chapters, a formal definition of the model is needed. Until now we discussed the concepts of this thesis colloquially on simple examples and without great detail. Furthermore, the formalization enables automated processing and analysis of the model by algorithms.

6.2.1 Formalizing Token Processing Networks

From the colloquial description given, the three terms *token*, *processing systems* and *graph* can be derived. With Equation (2.3) a formal construct for the term *token* was introduced. With this definition at hand we can

identify a set of systems $\mathcal{S}_\mathcal{K}$ that are available for coupling with other systems and furthermore exchange tokens:

$$\mathcal{S}_\mathcal{K} := \{S \mid S \in \mathcal{S}_{XY} \wedge \mathcal{K} \in \hat{X} \cup \hat{Y}\} \tag{6.1}$$

The coupling of the systems is described as:

$$\mathcal{C}_\mathcal{K} := \{(S_i, S_j, K_{ij}) \mid S_i, S_j \in \mathcal{S}_\mathcal{K} \wedge K_{ij} \subset \mathcal{K} \wedge K_{ij} = Y_i = X_j\}\,. \tag{6.2}$$

where K_{ij} denotes the subset of the output of S_i and the input of S_j, respectively. K_{ij} depicts the part that is used to couple the systems together (as described in Chapter 4.1.1). Elements of $\mathcal{C}_\mathcal{K}$ are commonly called *channels* (e.g. in [Inc09]). For convenience, the predecessors $\rightarrow_S$ and successors ${}_S\rightarrow$ of a system $S \in \mathcal{S}_\mathcal{K}$ are specified as

$$\begin{aligned} \rightarrow_S &:= \{S' \mid S' \in \mathcal{S}_\mathcal{K} \wedge (S', S, .) \in \mathcal{C}_\mathcal{K}\} \\ {}_S\rightarrow &:= \{S' \mid S' \in \mathcal{S}_\mathcal{K} \wedge (S, S', .) \in \mathcal{C}_\mathcal{K}\} \end{aligned}$$

In conclusion, a *material flow model* is defined by a triple

material flow model specification

$$(\mathcal{K}, \mathcal{S}_\mathcal{K}, \mathcal{C}_\mathcal{K}). \tag{6.3}$$

The *material flow graph* G of a given material flow model is defined as $G := (\mathcal{S}_\mathcal{K}, \mathcal{C}_\mathcal{K})$.

Example

The material flow graph $(\mathcal{S}, \mathcal{C})$ from Figure 4.4 is defined by the following two sets

$$\begin{aligned} \mathcal{S} &:= \{\text{Source}, \text{Truck}, \text{Warehouse}, \text{Production}, \text{Sink}\} \text{ and} \\ \mathcal{C} &:= \{(\text{Source}, \text{Truck}, \mathcal{K}), (\text{Truck}, \text{Warehouse}, \mathcal{K}), \\ &\quad\ (\text{Warehouse}, \text{Production}, \mathcal{K}), (\text{Production}, \text{Sink}, \mathcal{K})\}. \end{aligned}$$

This is a description technique found in almost every current enterprise simulation software. The current description does not provide any information about the behavior of the components $\mathcal{S}_\mathcal{K}$ used. The dynamic of the components will be formalized as little as possible, while the token states will be fully exposed for analysis and manipulation. This is done in the next chapter.

Token Processing Systems

In the colloquial definition of *material flow systems* given in Chapter 2.1.2 it was defined that a system works with tokens it owns. This means, it only has access to such tokens which it has created itself or were a part of its

input in the past but were not part of its output yet. Let $S \in \mathcal{S}_\mathcal{K}$ be an arbitrary *token processing* system and $t \in \mathcal{T}$ a point in time and $k \in \mathcal{K}$ an arbitrary token. *Processing* in this context means all sorts of actions that may be applied to tokens. This especially includes the delay and altering and storage of tokens. S has to store its tokens in some (physical) space. Let I be an index set, then $P_S \subset S$ defined as

$$P_S := \{p_i \mid i \in I\} \tag{6.4}$$

is a set of places where S can store tokens. Per definition, each place p_i can only store one token. Let $q_S : P_S \to \{\mathcal{K} \cup \emptyset\}$ be a function that returns for a given place $p_i \in S$ the stored token $k \in \mathcal{K}$. If no token is stored at p_i, q_S returns the empty set $\emptyset$. Then the set of occupied places $O_S := \{p_i \mid q_S(p_i) \in \mathcal{K}\}$ and the set of unoccupied places $\overline{O}_S := \{p_i \mid p_i \notin O_S\}$ can be specified. From the definition of O_S follows that $K_S = \bigcup_{p_i \in O_S} q_S(p_i)$. Now that we are able to express the local storage of tokens for processing within a system, we also have to describe whether a place (generally) can be accessed from external.

$q_S(.)$

Consider a simple shelf that stores some boxes with unknown content. Some of the places in the shelf are occupied by boxes (O_S) and some are free ($\overline{O}_S$). Because the shelf is tightly packed, places in the back of the shelf can be rendered inaccessible (blocked), due to boxes in the front. Another example are machines in general. Workpieces are stored in a work area for processing. This area is modeled as a storage place p_i. During the processing the area and the workpiece are usually not accessible from external, or at least the workpiece processing should not be interfered. Let $a_S : P_S \to \{\text{true}, \text{false}\}$ be a function, that describes whether a place $p_i \in P_S$ is accessible from external (*true*) or not (*false*). Then it is possible to specify the set of accessible places $A_S := \{p_i \mid a(p_i) = \text{true}\}$ and the set of inaccessible places $\overline{A}_S := \{p_i \mid a(p_i) = \text{false}\}$. The combination of these two attributes (occupation and accessibility) defines four sets of places as depicted in Figure 6.2.

$a_S(.)$

6.2.2 Material Flow System States

With the functions q_S and a_S in place, we are now able to describe the state U_S of a token processing system S as a sequence of the states $u(p_i)$ over all its places $p_i \in P_S$:

$$U_S := (u(p_0), u(p_1), \ldots) \text{ with } p_0, p_1, \ldots \in P_S$$

and

$$u(p) = \begin{cases} (q_S(p), a_S(p)) & \text{if } q_S(p) \in \mathcal{K}, \\ (\emptyset, a_S(p)) & \text{otherwise} \end{cases} \quad \text{where } p \in S. \tag{6.5}$$

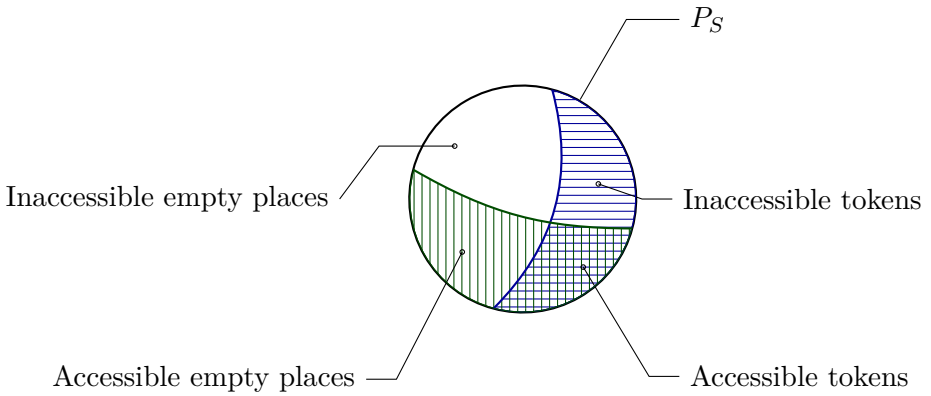


Figure 6.2: The structure of the local storage.

Let J be an index set with which we can specify an arbitrary but fixed sequence of the systems in $\mathcal{S}_\mathcal{K}$. J should not be mistaken for a definition on how the systems are connected, instead its just a fixed enumeration of the them. For convenience we will write S_j for the j-th system according to the sequence specified by J. Then the state U of the whole material flow model, is defined as the combination of the state of all systems:

$$U := (U_{S_0}, U_{S_1}, \ldots) \tag{6.6}$$

Additionally, with J we can define the following mapping

$$p_{ij} : I \times J \to \bigcup_{S \in \mathcal{S}_\mathcal{K}} P_S \tag{6.7}$$

where p_{ij} specifies the i-th place (according to Equation (6.4)) of the j-th system. p_{ij} can be used to identify each place in the material flow uniquely. Now with Equation (6.5) and (6.7) the state U can also be written as

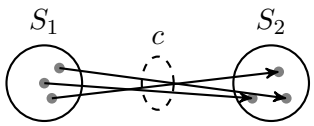
$$\begin{aligned} U := (\quad & (u(p_{00}), u(p_{10}), \ldots), \\ & (u(p_{01}), u(p_{11}), \ldots), \\ & \vdots \\ & (u(p_{0n}), u(p_{1n}), \ldots) \quad) \end{aligned} \tag{6.8}$$

where $n = |\mathcal{S}_\mathcal{K}|$ denotes the overall number of systems in the model. The state space $\mathcal{U}$ of a material flow model is then defined as

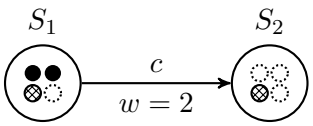
$$\mathcal{U} := (I \times J) \times (\{\mathcal{K} \cup \emptyset\} \times \{true, false\}) \tag{6.9}$$

Channel States

The channel definition from (6.2) does not specify any token transformation mechanism or state space. Both, the transformation and the states can be



(a) Channel c specifying a bijective mapping between the places of S_1, S_2.



(b) Channel c has a weight w of two. There is no specific mapping of places (inner circles).

Figure 6.3: Two different types of channels.

constructed from the state definition of the token places (6.5).

Let $c \in \mathcal{C}_\mathcal{K}$ be a channel in a material flow model according to Equation 6.2. By definition the tokens in $K_{ij} \subset \mathcal{K}$ will move from S_i to S_j through c at some point in time, but it is never explained *why* or *when*. To describe this mechanism, the notion of *enabled* channels is introduced. A channel is said to be *enabled*, when all its requirements are satisfied. For an *enabled* channel always a bijective mapping $m : P_{S_1} \to P_{S_2}$ between P_{S_1} and P_{S_2} is defined. Such a mapping can be completely random and may change without further notice. In the following two different types of channels, each having their own requirements are introduced. Let S_1, S_2 be two token processing systems, each with a set of places P_{S_1} and P_{S_2}, and let $c := (S_1, S_2, \mathcal{K})$ be a channel between those two systems:

Enabled Channels

1. The first type of channels is associated with a fixed bijective mapping $m : P_{S_1} \to P_{S_2}$. c is said to be *enabled* iff

$$\bigwedge_{(p,p') \in m} p \in O_{S_1} \cap A_{S_1} \wedge p' \in \overline{O}_{S_2} \cap A_{S_2}. \tag{6.10}$$

   This type of channels explicitly links pairs of places from S_1 and S_2. If in each of these places from S_1 resides an accessible token, and each place from S_2 is empty and associable, the channel is enabled. An example of such a channel with a mapping of size three is depicted in Figure 6.3a.

2. The second type of channels does not specify a fixed mapping, instead it has a weight w. Then c is said to be *enabled* iff

$$|O_{S_1} \cap A_{S_1}| \geq w \wedge |\overline{O}_{S_2} \cap A_{S_2}| \geq w \tag{6.11}$$

   In words, it is enabled, if S_1 provides access to at least w tokens, while S_2 at the same time provides enough accessible places to receive w tokens. The mapping m for this type is built ad hock and is unspecified and completely random. In Figure 6.3b such a channel with weight $w = 2$ is depicted as an example. Here, the channel is enabled as there are two tokens available (●) in S_1 while there are also two empty places (◌) in S_2. Due to (6.11) this channel appears to be *enabled*.

For convenience, we will identify the places in S_1, currently associated with a channel c as $\rightarrow_c$, and the places from S_2 with ${}_c\rightarrow$, respectively. If a place p is contained in either $\rightarrow_c$ or ${}_c\rightarrow$ then we may depict this simply with $p \in c$. Now we have everything together to describe the dynamical evolution of material flow models.

6.2.3 Material Flow Dynamics

Let $S \in \mathcal{S}_\mathcal{K}$ be a token processing system with a set of places P_S. Since S has access to its local token storage, it can make changes to the four sets $\{A_S, \overline{A_S}\} \times \{O_S, \overline{O_S}\}$. The possible actions are:

- Setting a place $p \in P_S$ accessible or inaccessible (Changing the value of $a_S(p)$).
- Putting a token $k \in \mathcal{K}$ into a specific place p, either by *creating* the token or by moving k to p from another place $p' \in P_S$.
- Remove a token from a place p, emptying the place (Setting $q_S(p) = \emptyset$).
- A combination of the above actions. For example, for the movement of a token from a place p to another one p', the token must be removed from p' and put into p.

However, one generic restriction does apply at any given point in time: For a token $k \in \mathcal{K}$ it is not allowed to be in two places at any given point in time, i.e.:

$$\bigwedge_{k\in\mathcal{K}} \bigvee_{i\in I} \bigvee_{j\in J} q(p_{ij}) = k \implies \bigwedge_{i'\in I\setminus\{i\}} \bigwedge_{j'\in J\setminus\{j\}} q(p_{i'j'}) \neq k$$

Channel Dynamics

Let $(\mathcal{S}, \mathcal{C}, \mathcal{K})$ be a material flow model. Furthermore, let $\mathring{\mathcal{C}} \subseteq \mathcal{C}$ be the set of *enabled* channels. The state transition function $f_\mathcal{C} : \mathcal{U} \times \mathring{\mathcal{C}} \to \mathcal{U}$ for material flow models is only defined for enabled channels. Different types of channels may have different requirements which must be satisfied for the channel to be *enabled*. However, one explicit condition, resulting from the semantic definition of a_S is the following, that all channels have to meet to be identified as *enabled*.

$$c \in \mathring{\mathcal{C}} \implies \bigwedge_{p\in c} a(p) = true \wedge \bigwedge_{p\in\rightarrow_c} q(p) \in \mathcal{K} \wedge \bigwedge_{p\in{}_c\rightarrow} q(p) = \emptyset \tag{6.12}$$

We refer to the carrying out of a state transition based on an enabled *Triggering a Channel*

channel, as *triggering* this channel.

$$\dot{u}(p_{ij}) = \begin{cases} (q(p'), \text{true}) & \bigvee_{c \in \mathring{\mathcal{C}}} (p', p_{ij}) \in c \\ (\emptyset, \text{true}) & \bigvee_{c \in \mathring{\mathcal{C}}} p_{ij} \in \rightarrow_c \end{cases} \tag{6.13}$$

With Equation (6.12) it is ensured that only those channels are enabled, where the involved places are accessible from external. This requirement is needed, as channels are systems for themselves, that do access (and manipulate) the places of a system from external. Furthermore, we require, that all involved places in system S_1 contain a token, while all places in system S_2 have to be empty. This is required, so that the state transition according to Equation (6.13) does not overwrite tokens in S_2. If a channel is *triggered*, according to Equation (6.13), the tokens, residing in the places from S_1 are transferred to S_2. This transfer is coordinated by the channels mapping m_c. The accessibility state of all involved places remains unaffected. With the Equations (6.10) and (6.11) two types of channels were introduced. Each defining the mapping m_c differently. Other types of channels with arbitrary requirements are imaginable.

Furthermore, because of Equation (6.12) a system S can control the movement of tokens by controlling the accessibility of its places P_S, especially those, referenced by a channel. By blocking all places it is not possible for new tokens to arrive. By setting places with tokens accessible, these tokens may move on to the next system at any time.

Example *The Facility*

In the following the *facility* system from earlier (see Figure 4.4) is reviewed, utilizing the structures introduced in the previous chapter. The different blocks behave as one would expect, e.g. the *Source* occasionally generates new tokens, while the *Truck* moves from a (virtual) start location to a (virtual) destination delivering tokens. The warehouse stores tokens for the production, which can process two tokens in parallel. The model starts with the configuration shown in Figure 6.4a. The circles show the states of their distinct places. All channels are of type *two* (cp. Equation 6.11). The numbers at the channels depict their weight.

The *Source* contains a token that is available (●) while the *Truck* has an empty place (◌). Therefore, the channel between the *Source* and the *Truck* is *enabled*. However, the *Truck* is currently located at the *Source*. Therefore the token contained in the *Truck* is blocked (⊗). Furthermore, because Equation 6.11 is not fulfilled, the channel between the *Truck* and the *Warehouse* is *disabled*. The token contained in the *Production* is currently processed by a machine. Therefore it is also blocked.

Consider we moved forward in time, reaching the state depicted in Figure 6.4b. Here the token from the *Source* was loaded onto the *Truck*. Because

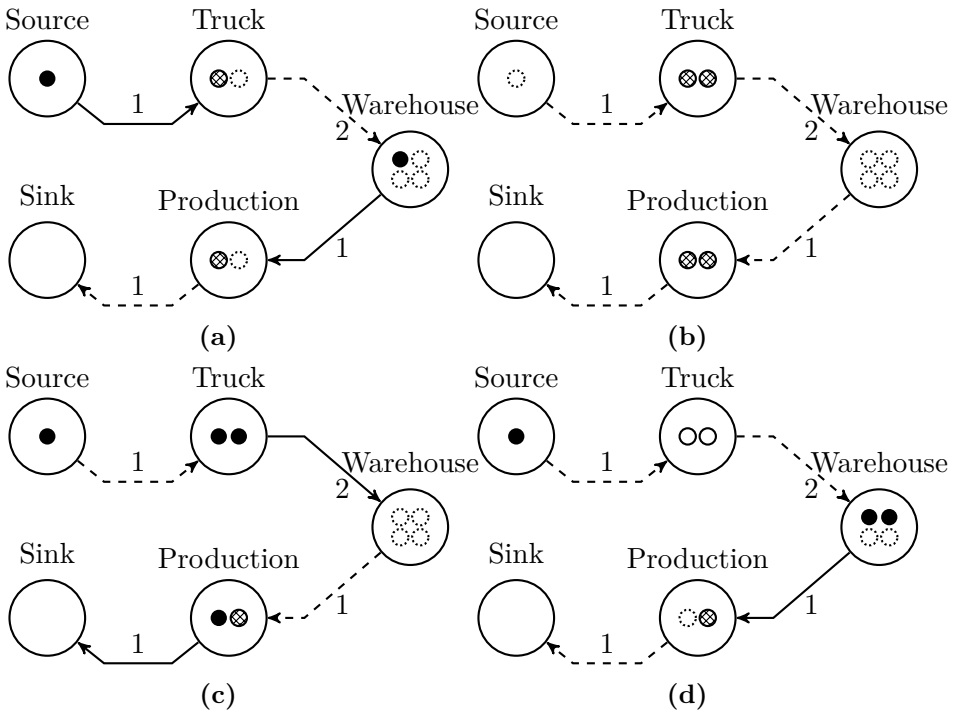


Figure 6.4: State of the facility example at different points in time. The tokens are represented by the different circles. ● $\in A_S \cap O_S$ (Accessible Token); ⊗ $\in \overline{A}_S \cap O_S$ (Blocked Token); ◌ $\in A_S \cap \overline{O}_S$ (Empty, Accessible Space); ○ $\in \overline{A}_S \cap \overline{O}_S$ (Empty, Blocked Space). Solid lines depict *enabled* channels, while dotted lines are used for *disabled* channels.

it is fully loaded, the *Truck* is on its way to the *Warehouse*. It has not reached the *Warehouse* yet, therefore both tokens are blocked. The last token was moved from the *Warehouse* into *Production*, leaving the *Warehouse* completely empty (◌). Still, the first token in *Production* has not reached the end.

At the next point in time shown in Figure 6.4c the *Truck* has reached its destination. The tokens are now accessible and can be moved into the *Warehouse*. Equation 6.11 is fulfilled and the tokens are moved from the *Truck* into the *Warehouse* (cp. Figure 6.4d). However, the first token from *Production* is finished and is now ready to move into the *Sink*.

In Figure 6.4d the *Truck* has been unloaded. Now the *Truck* has to drive back to the *Source* so that he can be loaded again. During this process, tokens from the *Source* may not be loaded onto the *Truck*. Therefore, its places are blocked (○).

6.2.4 Implementation as a Discrete Event System

In the example *The Facility* we implicitly used specific points in time where the state of the model changed. The state change mechanism was never explicitly defined, which is what will be done in this section. The discrete nature of the state space $\mathcal{U}$ (cp. Equation (6.9)) is well suited for the implementation as a *Discrete Event System* (DES). As specified in Chapter 2.2.2 a DES changes its state through executed events. In the following a set of observable events will be defined, that describe state changes of a material flow model.

Equation (6.8) clearly describes U as a combination of all states of all places. Therefore, the state change of a sole place changes the state of the whole model. That means, the following set of four events describes every possible state change.

$$E := \{\check{a}_p, \check{\overline{a}}_p, \check{q}_p, \check{\overline{q}}_p\} \tag{6.14}$$

where

$\check{a}_p$ is the change of place p from *blocking* to *accessible*

$$p \in \overline{A} \quad \xrightarrow{\check{a}_p} \quad p \in A,$$

$\check{\overline{a}}_p$ is the change of place p from *accessible* to *blocking*

$$p \in A \quad \xrightarrow{\check{\overline{a}}_p} \quad p \in \overline{A},$$

$\check{q}_p$ adds a token $k \in \mathcal{K}$ to the *empty* place p

$$q(p) = \emptyset \quad \xrightarrow{\check{q}_p} \quad q(p) = k,$$

$\check{\breve{q}}_p$ removes a token $k \in \mathcal{K}$ from place p

$$q(p) = k \quad \xrightarrow{\check{\breve{q}}_p} \quad q(p) = \emptyset.$$

Please note, that this set is not the only possible description. For example, $\check{q}_p$ and $\check{\breve{q}}_p$ miss the fact that they can be triggered by a channel transition (6.13) or by an internal mechanisms of S (where $p \in S$). Other sets describing different aspects of the state change could also be specified. We will do this later on, when we want to analyze the performance of token processing system.

6.2.5 Concluding Remarks

To allow the performance analysis of a token processing system, specific structures where introduced, such as *places* and *channels*. *Places* allow the observation and analysis of the processing of tokens that take place within a system. *channels* allow the observation of the token movement between systems. Because of this approach, the token processing systems and their behavior do not have to be specified in any way. Therefore, the specification offers a deep view into the internals of a material flow system, without restricting the model to certain well-known components.

Please note, that the material flow system in a model usually is only a part of a larger model. The model may contain additional systems from other domains. These subsystems may even be connected somehow to components in the material flow system and influence or control their behavior.

6.3 Performance of a Token Processing System

The model specifications introduced in Chapter 6.2 have been described from a place centric perspective. Especially, the event set defined in (6.14) describes the behavior of places. To analyze the behavior of tokens residing in a token processing system (TPS), a modified event set must be derived that describes token state changes.

Event Sequence of a TPS

Given is an arbitrary TPS S. During simulation of S a sequence $\check{E}_{i_S}$ of events can be observed (with $\{\check{E}_{i_S}\} = E$, where E is the set from Equation 6.14). The sequence $\check{E}_{i_S}$ is place-centric, i.e. with $\check{E}_{i_S}$ it is easily possible to describe the behavior of a certain place, since each event in $\check{E}_{i_S}$ belongs to a certain place. However, to examine the behavior of a certain token $\check{E}_{i_S}$ must be partitioned in token related event sequences $\check{E}_{i_k}$. This is done by associating the events of $\check{E}_{i_S}$ to the token that is affected. Thus creating for each token k its own event sequence $\check{E}_{i_k}$:

Event Sequence of a Token in a TPS

$$\check{E}_{i_k} \subseteq \check{E}_{i_S} := \{\check{e} \mid \check{e} \in \check{E}_{i_S} \wedge \check{q}(\check{e}) = k\} \tag{6.15}$$

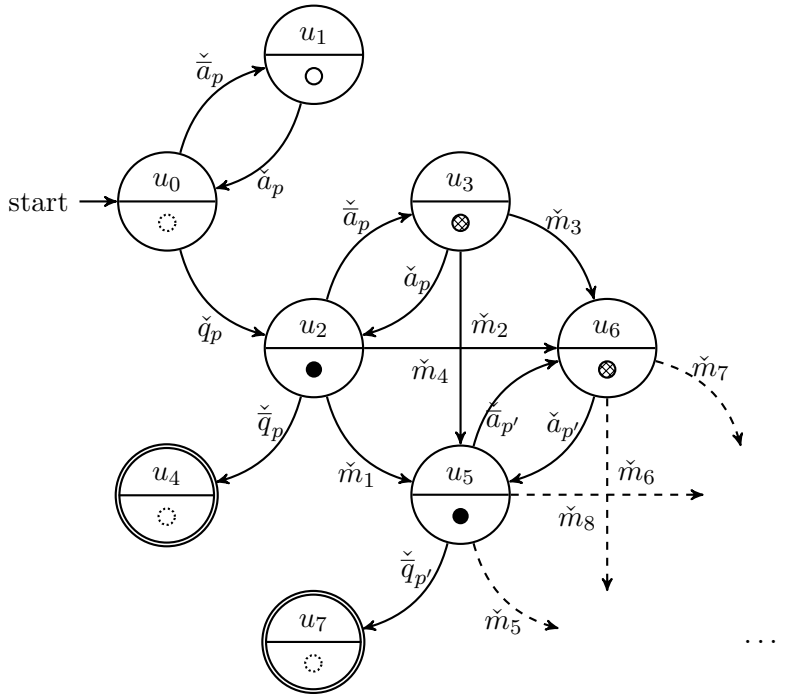


Figure 6.5: State machine describing the state of a specific token during its stay in a system. Each state has a name (u_i) and a token state, indicated through the circle in the lower part of the state. The depicted events are either from Equation (6.15) or indicate the movement from one place to another ($\breve{m} := (\breve{\breve{q}}_p, \breve{q}_p) \wedge \breve{q}(\breve{\breve{q}}_p) = \breve{q}(\breve{q}_p)$). *Back movements are not depicted to keep the overview.*

where $\breve{q}()$ is a function that returns the affected token for a specific event $\breve{e}$. $\breve{q}()$ utilizes the function $q()$ as well as the fact that every event $\breve{e} \in E$ (6.14) is associated with a specific place p. In the following, let $\breve{e}_i, \breve{e}_j, \breve{e}_h \in \breve{E}_{i_S}$ be three events and $p(\breve{e})$ the by $\breve{e}$ affected place, then $\breve{q}()$ is defined as follows:

$$\breve{q}(\breve{e}_i) = \begin{cases} q(p(\breve{e}_i)) & \text{if } \breve{e}_i \in \{\breve{a}_p, \breve{\breve{a}}_p, \breve{q}_p\} \wedge q(p(\breve{e}_i)) \neq \emptyset \\ \breve{q}(\breve{e}_{i-1}) & \text{if } \breve{e}_i = \breve{\breve{q}}_p \\ \breve{q}(\breve{e}_j) & \text{with } i \leq h < j \wedge \breve{e}_h \neq \breve{q}_p \wedge \breve{e}_j = \breve{q}_p \\ & \text{if } \breve{e}_i \in \{\breve{a}_p, \breve{\breve{a}}_p\} \wedge q(p(\breve{e}_i)) = \emptyset \end{cases}$$

Figure 6.5 depicts a universal state machine for arbitrary TPS, describing all possibly observable $\breve{E}_{i_k}$. The events that lead to a state change are depicted along the edges, connecting the different states. Let $k \in \mathcal{K}$ be a token that arrives at S. Furthermore, let p be the place where k is stored. Then p must be empty and accessible before k can arrive (u_0 in Figure 6.5). By setting p inaccessible beforehand (u_1) and making it accessible again at

a certain point in time, S can control the arrival of k. We can think of this process as *preparation*, e.g. because of the *retooling* of a machine or a *cleanup* of p. At some point in time, k arrives, indicated by $\check{q}_p$. In this state (u_2) k can leave S at any moment (u_4) since p is accessible. Again, to control the point in time p can be blocked for access (u_3). Since the places contained in P_S are distinguishable from each other, S may use this to structure its tokens. Therefore, S may move k from one place (here p) to another (here $p' \in S$) at any point in time (u_5 and u_6). This movement is indicated by the events depicted as $\check{m}_i$. Basically, they are the combination of two events indicating the removal (from p) and the deposition of k (into p'). For $\check{m}_1$ and $\check{m}_2$ these are $\check{\overline{q}}_p$ and $\check{q}_{p'}$. Obviously, for $\check{m}_3$ and $\check{m}_4$ these are $\check{\overline{q}}_{p'}$ and $\check{q}_{p''}$. This scheme continues for all places in S.

For each token k that arrives at S it is now possible to specify a sequence of states u_i. If S has an infinite set of places, this state machine also has an infinite set of states. Even for a system with limited capacity sequences from different tokens will not have much in common. Therefore, behavior observations in the form of these sequences will be hard to compare and to analyze. For example, from u_0 the machine may switch to every other state except u_1 and u_{4+3i} with $i = 0, 1, \ldots$. In the following, the number of different states will be reduced. This is achieved by

1. omitting the token movement, i.e. omitting all $\check{m}_i$ events. That means, different places and the associated events are no longer distinguishable. For example, the states $u_2, u_5, \ldots$, where the token is accessible, can be merged into a single state ●.

2. TPS where tokens will be at least once in state ⊗ are named *machines*. It is assumed that tokens become inaccessible as soon as they enter the machine (◌ $\rightarrow \check{q}, \check{\overline{a}} \rightarrow$ ⊗). Otherwise, the tokens would be accessible (●) for a period of time and could be pulled into a subsequent TPS, skipping the processing altogether. For a full specification an external clock is needed. This is discussed in the next section.

3. Simple storage systems like buffer can omit the states v and b. Their state machine is reduced to the remaining three states l and w and f. Therefore, for these special TPS the following holds: $\bigwedge_k \bigwedge_{e_i \in \check{E}_{i_k}} e_i \notin \{\check{\overline{a}}, \check{a}\}$.

4. Later on, we will see that the preprocessing step v is insignificant for the coarsening method. This is due to the fact, that the time tokens are waiting for the preprocessing to finish is already absorbed in the waiting state w of a previous system. Therefore, it will be omitted in later chapters.

Under the described assumptions the universal state machine (partially) shown in Figure 6.5 collapses into the state machine with five states depicted

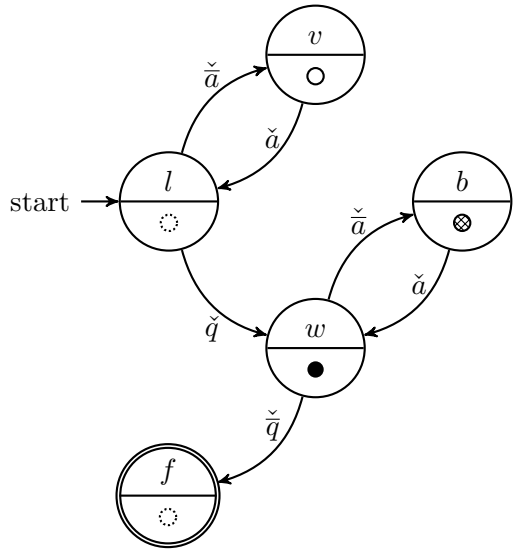


Figure 6.6: Finite state machine depicting the reduced state set. Since places are no longer unique, the parameter p is omitted for the events.

in Figure 6.6. The states are named after their semantical meaning. v can be used to simulate preparations taking place before a token arrives (*preprocessing* state). l is the *idle* state, as the TPS is waiting for the token to arrive. With setting the arrived token inaccessible, a system indicates that the processing of a token is about to start. Therefore, b is the *processing* state. If the token resides in w it is waiting for further processing or moving to the next system. Therefore, this is the *waiting* state. f identifies the state where the token has left the system (*accepting* state).

6.3.1 Adding an External Clock

Until now the concept of time was completely omitted. Sequences of events were observed without any reference to a clock. Let $\mathcal{T}$ be a clock structure (cp. Chapter 2.2.2). Now each event $\check{e}_i \in \check{E}_{i_k}$ can be assigned a point in time $t_{\check{e}_i} \in \mathcal{T}$ where it occurred. That makes it possible to describe, how long a token stayed in the different states ○, ⊗ and ●.

$$
\begin{aligned}
\circ_k &= \begin{cases} 0 & \text{if } \check{e}_0 \neq \breve{a}, \\ t_{\check{e}_j} - t_{\check{e}_0} & \text{where } \check{e}_j = \check{a} \wedge \check{e}_{j+1} = \check{q} \wedge \check{e}_{j+2} = \breve{a} \text{ with } t_{\check{e}_{j+1}} = t_{\check{e}_{j+2}} \end{cases} \\
\otimes_k &= \begin{cases} 0 & \text{if } \bigwedge_{\check{e}_j \in \check{E}_i} \check{e}_j = \check{q}_p \Rightarrow \check{e}_{j+1} = \breve{q}, \\ t_{\check{e}_j} - t_{\check{e}_i} & \text{where } \check{e}_i = \check{q} \wedge \check{e}_{i+1} = \breve{a} \wedge \check{e}_j = \check{a} \wedge \check{e}_{j+1} = \breve{q}, \end{cases} \\
\bullet_k &= t_{e_{j+1}} - t_{e_j} \quad \text{where } e_j \in \{\check{q}, \check{a}\} \wedge e_{j+1} = \breve{q}.
\end{aligned}
\tag{6.16}
$$

$\circ_k$ is obvious defined as the time from the first start of the preprocessing until the point in time of the arrival of k. $\otimes_k$ on the other side also collects time a token states in $\bullet$ between two visits to $\otimes$. An example scenario would be a production line where a metal part must have a specific minimum temperature when leaving, thus forcing the production line to reheat (reprocess) the part when a blockage occurs. $\bullet_k$ is defined as the time a token stays in the waiting state before it is pulled into a subsequent system.

6.3.2 Sampling a TPS

Let S be a token processing system. During a simulation run a sequence of tokens $k_0, k_1, \ldots$ passes through S. For each token k an event sequence $\check{E}_{i_k}$ can be observed from which $\circ_k$, $\otimes_k$, $\bullet_k$ can be computed (6.16). The coarsening method presented later on uses samples of these values to imitate the behavior of the TPS. The values are stored in look-up tables of fixed size. Usually, a new value that is added to a table overwrites the oldest entry. In the following the look-up tables will be depicted with [.]. For example, $[\otimes]_S$ specifies the look-up table for token processing times for system S.

Look-up Tables

However, the preprocessing time $\circ$ will not be sampled. This is due to the fact, that the time tokens are waiting for the preprocessing to finish is already absorbed in the waiting state of a previous system. Imagine two connected TPS S_1 and S_2, where S_2 does a full cleanup after each processed token, forcing subsequent token to wait in S_1. Now two cases have to be examined:

- S_1 is empty. In this case the preprocessing does not affect any token. Recording this time and using it for coarsening would impair the result.
- S_1 is not empty and a token k is waiting to move on to S_2. In this case the cleanup process *does* affect the simulation outcome - or at least the processing of k. However, the time for which k is waiting for S_2 to finish its cleanup is already included in $\bullet_k$. Therefore, the preprocessing time must not be recorded.

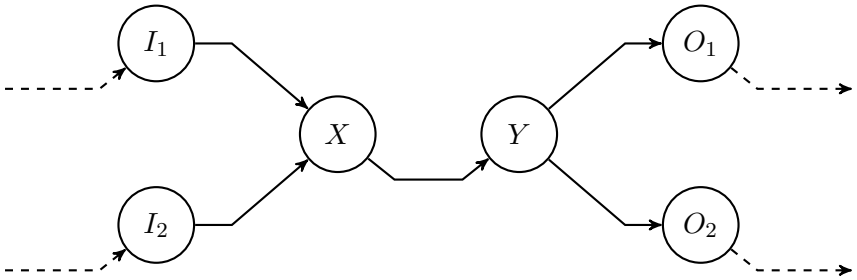


Figure 6.7: Imagine this structure as part of a larger material flow graph.

Because in both cases, the preprocessing time is of no use for the coarsening process it is omitted in later chapters.

With the state machine from Figure 6.6 and the equations from (6.16) a system has been defined that samples key data of token processing systems. The samples are taken at runtime for each token processing system in the simulation model. They depict the performance of a token processing system at a specific point in time. Now the samples can be used to parametrize replacement components that imitate the behavior of the token processing systems. For the coarsening method to be more efficient, not a single TPS is replacement but a whole group of TPS. In the following chapter a method is introduced to identify groups of TPS that can be coarsened by this concept.

6.4 Identifying Groups of Systems for Coarsening

The coarsening concept presented in this thesis can coarsen regions of the material flow graph that have exactly one entry and one exit component. That means, there is exactly one component that has incoming edges from components not in the group and there is analogous one component that has outgoing edges to external components. Furthermore, for every component within the group there must exist at least one path from the entry component and at least one path to the exit component (no dead ends and sinks or sources are allowed within such a group). Other than that, the connection structure in-between can be arbitrary. Such groups are called *single-entry-single-exit* (SESE) *regions* (cp. Chapter 4.4.3).

These restrictions help to keep the overall concept simple and robust and valid for general material flow systems. Let's briefly explore a counterexample: Assume that you have a material flow model with the graph depicted in Figure 6.7 being a part of it. Furthermore, we already have decided that X and Y should be replaced by a simplified component. Now let's examine the number of additional options on simplifying this structure. Since we allow any arbitrary (but connected) group of components, we can consider the following groups:

$$\{X, Y\} \cup \mathcal{P}\left(\{I_1, I_2, O_1, O_2\}\right),$$

where $\mathcal{P}$ refers to the power set. This means, we would have to consider sixteen different groups. Unfortunately, this problem exponentiates with the size and connection degree of the graph. Additionally, the replacement component would have to deal with several incoming and outgoing edges at various locations and the possible paths between them. Instead the groups that can be coarsened are restricted to SESE regions. This has the advantage, that the region can be reduced to a simple input-output-system (S, X, Y) where the entry node is X and the exit node is Y. Furthermore, every token that enters the system must also leave the system at some point (not necessarily in the same order they enter). This makes it much more easier to implement the replacement components. To find all SESE regions within the material flow graph a modified version of the *Program Structure Tree* by Johnson et al. is used.

6.4.1 Modified *Program Structure Tree*

The original algorithm can identify edge-based SESE regions in control flow graphs (e.g. of computer programs), it was altered in such a way that it can identify SESE nodes instead of SESE edges. For each SESE region of the PST we simply choose the destination node of the entry edge as the entry node and the source node of the exit edge as exit node of our new SESE region. It is easy to see that this does not violate the conditions for SESE regions. In a post-processing step nodes representing regions with only a single node are removed. Coarsening such a region does not provide any speed-up in practice. The only problem that remains is, that the PST algorithm can only handle graphs with a single entry node and a single exit node. For general material flow graphs we have to consider cases with multiple sources and sinks. To solve this problem we can simply add a *super source* that is connected to all sources of the MFS graph and a *super sink* that is connected to all sinks of the MFS graph. These two artificial nodes can be removed after the algorithm has been executed.

6.4.2 Dynamics

The tree can be updated upon insertion or deletion of nodes. This allows the whole concept to adapt itself to changes made on the material flow graph at runtime. Fortunately, we don't have to update the SESE regions upon each node insertion or deletion. When there is no violation of the SESE region conditions, we don't have to take any further steps. For example, when we extend a sequential part of a SESE region, the region stays valid and does not have to be recomputed. When the SESE region is not valid, we have to identify the smallest region that contains the changes and does not violate

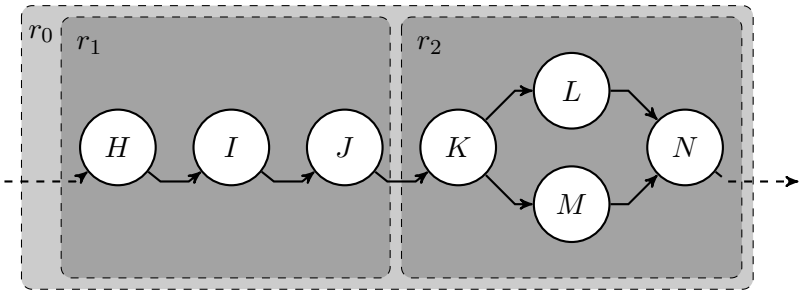


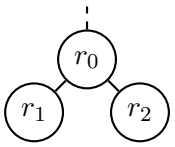
Figure 6.8: An abstract material flow example. Three SESE regions a, b, c have been identified (dashed rectangles).

the SESE region conditions. To recompute the SESE region, we simply need to modify the required depth-first search steps of the PST algorithm to not search the whole graph but only the sub-graph defined by the SESE region.

Although this can be easily implemented, our experiments showed that in most cases it is no problem to rebuild the entire tree of the complete graph. The algorithm only needs $\mathcal{O}(E)$ time, because it is based on a few depth-first searches. Therefore, for reasonable material flow graphs, there is no high benefit from running the dynamic PST algorithm.

6.4.3 Example

Figure 6.8 shows a small part of a material flow graph as an example. The dashed rectangles represent the identified SESE regions. The PST algorithm constructs a hierarchy from the identified regions. The hierarchy for the the example is depicted to the right.



Each of the identified regions r_0, r_1, r_2 can be coarsened with the concept presented in this thesis. It is possible to coarsen r_1 and r_2 independently from each other, as well as both or even to coarsen the whole structure, identified as r_0. An even larger example can be found in Appendix C.

In the following two chapters the construction of replacement components for the identified regions is discussed. First, the coarsening of sequentially connected regions like region r_1 from the example is presented. Afterwards in Chapter 6.6 a concept to coarsen arbitrary regions such as r_2 and r_0 is introduced.

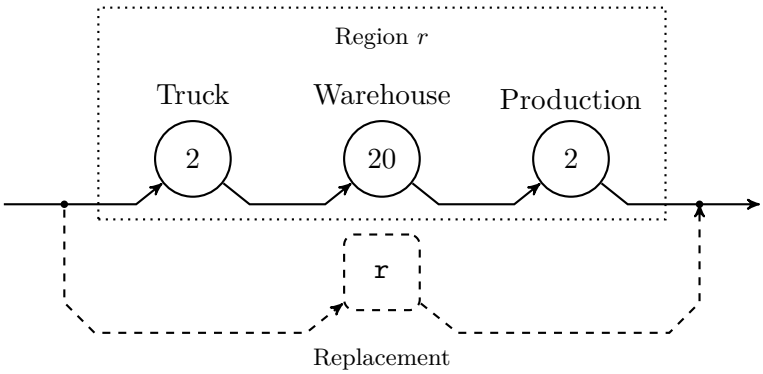


Figure 6.9: The region r will be replaced by `r`.

6.5 Coarsening Sequentially Connected Systems

Let $r := (S_0, S_1, S_2, \ldots)$ be a SESE region of the material flow graph where the contained systems $S \in r$ are connected sequentially, like the one shown in Figure 6.9. In the following, the necessary steps that are needed to coarsen r will be discussed. The goal is to replace r by a single system `r` that *imitates* the behavior of r. `r` needs less time for computation but usually also produces a different output than the original region r. The difference is a result of the different computations: While the output of r is computed by the the original components, `r` is a single component and must rely on the samples taken earlier. Therefore, after some time it must be possible to switch back to r to gain new samples.

The biggest part is the state construction (or state transfer) when switching from r to `r` and back. As outlined in Chapter 6.1 switching without a state transfer *restarts* the affected model part with an empty state. Much information gets lost and a huge error is introduced into the simulation results. Particularly due to their detailed knowledge of the behavior of the model components, Huber and Dangelmaier [HD09] are able to define a precise state transfer for each component. Unfortunately, in the presented model specification the definition of the internal processes were left undefined. Detailed knowledge about the material flow processes is not available and cannot be used. Then again: Due to omitting such knowledge arbitrary material flow processes are supported. The state transfer will be computed from previously observed event sequences and performance ratios for tokens passing through r.

In the next section needed information for a state transfer will be gathered. The switch can occur in both directions: From r to `r` and the other way round. Both directions will be discussed in the following sections. In the last section additional information is used to handle processes like the altering and assembly and disassembly of tokens.

6.5.1 Sampling r

Since all tokens passing through r follow the same path, sampling sequentially connected TPS is pretty easy compared to the coarsening of arbitrarily connected TPS. That means, each token is processed by the same systems in the same order. In the most simple case r just delays incoming token for a specific time. Usually, the delay comes from TPS' that actively delay tokens and also from the interactions between different TPS (bottleneck situations) which can create waiting times for the tokens. Therefore, the information that is needed to imitate r is the overall *lead time* of the tokens passing through r. Instead of sampling this value, it can be computed when needed from the look-up tables defined earlier:

$$[lt]_r = \sum_{i=1}^{n-1} \Big([\otimes]_{S_i} + [\bullet]_{S_i} \Big) + [\otimes]_{S_n} \tag{6.17}$$

where S_i specifies the i-th system in r. Note, that the waiting time of the last system $S_n \in r$ is not included. If a token has to wait after S_n has processed it, there must be a subsequent bottleneck. S_n is not responsible for this waiting time nor does it have any influence on it. Therefore, $[\bullet]_{S_n}$ is excluded.

6.5.2 Switching to the Coarsened Version ($r \rightarrow \mathtt{r}$)

A given material flow system has been simulated for some time. Samples of the different TPS have been taken (as described in Chapter 6.3). Due to a given metric a specific r has been identified for being a good candidate for coarsening. At this point in time a controlling process initiates the switch from the original group of systems in r to a replacement `r`. `r` itself is implemented as a component that delays incoming tokens for a specific time. It delays incoming tokens independently from each other and can contain $|r|$ tokens at its maximum, where $|r|$ is defined as

$$|r| = \sum_{S_i \in r} |P_{S_i}|. \tag{6.18}$$

Because `r` delays the tokens for nearly as long as r would, it delays them in parallel and independently from each other. `r` is used for a short period of time only so that the behavior of r in changing situations must not be predicted but simply can be resampled. To initiate `r` properly, every token $k \in r$ must be transferred to `r`. On transfer, for each k a point in time t_k must be predicted when r *would have* finished the processing of k. This is done by choosing a sample from $[lt]_r$ and adding to it the entry time of k into r. Given $[lt]_r$ and the event sequence $\check{E}_{i_k}$ for k, t_k is defined as

$$t_k = t_{e_0} + [lt]_r \text{ with } e_0 \in \check{E}_{i_k} \tag{6.19}$$

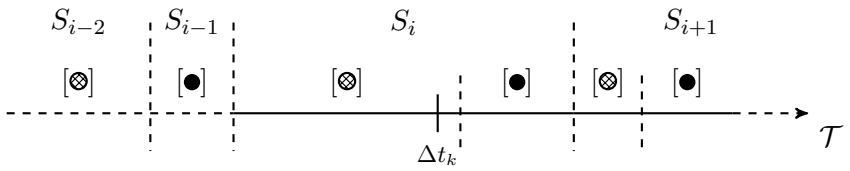


Figure 6.10: The systems contained in r can be outlined along a time set $\mathcal{T}$. Each system occupies space on the timeline based on the samples from its look-up tables. Each tick represents a point in time where a token can be put back into the time line or more accurately the structure of r.

where $[lt]$ depicts a random value from the look-up table constructed in Equation 6.17. t_k can be seen as a well informed guess until which point in time k would have stayed in r. While $\mathtt{r}$ is active new tokens may arrive, these are also delayed for a random value of $[lt]$.

6.5.3 Switch Back to the Original Version ($\mathtt{r} \rightarrow r$)

In the previous section a replacement component $\mathtt{r}$ for a region r was constructed and a state transfer was done to properly initialize $\mathtt{r}$. At some point later in time, a switch back to r is initiated to take new samples of the original components. As aforementioned, to avoid activating r without any tokens - which equals a restart - an initial state is computed by transferring the tokens from $\mathtt{r}$ back to r. Given is a token $k \in \mathtt{r}$ then a system $S_i \in r$ must be found to which k should be transferred. Basically, there are two option where k can be put:

⊗) k can be introduced as a new token that needs to be processed by S_i.

●) k can be added to S_i in such a way that it looks like it was already processed by S_i and now is waiting to move on to a subsequent TPS.

With the given sequence of TPS $(S_0, S_1, S_2, \ldots)$ in r for each option a point in time can be computed from the look-up tables from the different TPS. For example, the first option $\otimes_{S_0}$ is associated with t_0. The second option $\bullet_{S_0}$ is associated with $t_0 + [\otimes]_{S_0}$. The third option $\otimes_{S_1}$ then with $t_0 + [\otimes]_{S_0} + [\bullet]_{S_0}$ - and so on. This system is depicted in Figure 6.10. Each tick on the timeline is an option as specified before. The interval between two ticks represents the time it takes for a specific system to process a token ($[\otimes]$) or how long a token has to wait at a system ($[\bullet]$) - according to the samples from the look-up tables for each system. When given a specific token k that has to be put back into r an appropriate option must be found.

Let $\Delta t_k = t - t_{\check{e}_0}$ with $\check{e}_0 \in \check{E}_{i_k}$ be the time k was delayed until the switch was triggered (i.e. t is the current point in time). Then the nearest option relative to Δt_k is chosen for k. In the example from Figure 6.10 the

chosen option would be $\bullet_{S_i}$. The equations to determine the system S_i and the specific option look as follows:

$$\begin{aligned} &\sum_{j=0}^{i-1}\left([\otimes]_{S_j}+[\bullet]_{S_j}\right) \leq \Delta t_k < \sum_{j=0}^{i-1}\left([\otimes]_{S_j}+[\bullet]_{S_j}\right)+\frac{1}{2}*[\otimes]_{S_i} \Rightarrow \otimes_{S_i} \\ &\sum_{j=0}^{i-1}\left([\otimes]_{S_j}+[\bullet]_{S_j}\right)+\frac{1}{2}*[\otimes]_{S_i} \leq \Delta t_k < \sum_{j=0}^{i}\left([\otimes]_{S_j}+[\bullet]_{S_j}\right) \Rightarrow \bullet_{S_i} \end{aligned} \tag{6.20}$$

Of course, there are different types of TPS. For example, a simple buffer does (per definition) no processing of a token. Thus, its look-up table for $\otimes$ times will be empty. In such a case, the specific table in Equation 6.20 is simply replaced by a value of zero. In Figure 6.10 this case is depicted for system S_{i-1} which has a waiting time interval only. S_{i-2} in the same figure depicts the complementary case where no waiting time is given. Such a case appears for machines that have no waiting time, for example because the subsequent system (here S_{i-1}) is an infinite sized buffer that takes all processed tokens.

6.5.4 Handling Altering, Assembly, Disassembly of Tokens

Especially machines in a production facility often process to tokens passing through in a destructive way (i.e. during processing the original token is replaced). The coarsening concept can be adapted to support such processing options. In this section the options and required changes are discussed.

Altering A TPS S is said to alter a token $k \in S$ if it replaces or transforms it into another token $k' \in \mathcal{K}$. The altering process applied by S_i can be specified as a injective function $f_{S_i} : \mathcal{K} \rightarrow \mathcal{K}$. Based on f_{S_i} it is possible to define the subsequence of token altering systems $r_f \subseteq r$. Each system $S \in r_f$ uses an individual function f_S to alter tokens passing through. To support such functions in the coarsening method two changes have to be made:

First, when $\mathtt{r}$ is active, k must be altered based on the specifications by r when processing is finished ($\otimes_{\mathtt{r}} \rightarrow \bullet_{\mathtt{r}}$). There are two different cases: Either k was transferred during a switch or it arrived as a new token while $\mathtt{r}$ was active. In the first case k was transferred from a specific TPS $S_i \in r$. Then all subsequent altering processes $f_{S_j} \in r$ with $i \geq j$ must be applied. In the second case all altering functions of all TPS in r are applied.

Second, on the switch $\mathtt{r} \rightarrow r$ k has to be altered before it is transferred. Let $S_o \in r$ be the system where k will be put according to (6.20). Then all altering processes prior to S_o have to been applied. That includes all f_{S_i} with $S_i \in r$ and a) $i < o$ for the case $\otimes_{S_k}$ or b) $i \leq o$ for the case $\bullet_{S_k}$.

Assembly and Disassembly A system S is said to assemble tokens, if it replaces a set of tokens by a single new one. For example, a raft can be constructed from several tree trunks and a robe. Disassembly specifies the counterpart: A token is split up into several new ones. Like the alteration, also assembly and disassembly of tokens are special cases of a generic transformation where a set of n tokens is transformed into a set of m other tokens. The generic transformation with a fixed ratio $n : m$ is specified by a quintuple $(\mathcal{A}, \mathcal{B}, n, m, g)$ where

- $\mathcal{A} \subset \mathcal{P}(\mathcal{K})$ is a set of token sets,
- $\mathcal{B} \subset \mathcal{P}(\mathcal{K})$ is a set of token sets,
- n is a fixed number of tokens that is needed as input, i.e. $\bigwedge_{A \in \mathcal{A}} \|A\| = n$,
- m is a fixed number of tokens that is output, i.e. $\bigwedge_{B \in \mathcal{B}} \|B\| = m$ and
- g is a function $g : \mathcal{A} \to \mathcal{B}$.

Then alteration is a transformation with a fixed ratio of $1 : 1$ and assembly has a ratio of $n : 1$ and disassembly a ratio of $1 : m$ Given is a sequence of connected TPS in a region r that perform specific token transformations. By chaining the transformations together it is possible to compute a transformation between two systems that are not directly connected. This works like the speed transformation of a gear mechanism in a car. A token that enters the sequence of systems is like the rotation for one teeth on the drive gear (or input gear). Through transformation along the different gear ratios the driven gear (or output gear) will also rotate for a specific amount. This amount then can be seen as a number of tokens that are output by last TPS in a sequence. Chaining several transformations $g_{S_0}, g_{S_1}, \ldots$ together is pretty simple, given their ratios $n : m_{g_i}$.

$$\frac{m}{n}_r = \prod_i \frac{m}{n}_{g_i} \tag{6.21}$$

The transformations must be done on the switch $r \to \mathtt{r}$, during the active time period of $\mathtt{r}$ and on the switch $\mathtt{r} \to r$. Of course it is possible, that the output is not in whole numbers. Then the fraction is stored as a carry over and settled into the calculation for the next incoming token.

Restrictions

This system has some restrictions: During transformation information about which token contributed to which output is lost. That means, this compact description of the assembly/disassembly behavior of a region r works only with undistinguishable tokens. Using it together with altering system described earlier is *not possible*. To support assembly/disassembly together with

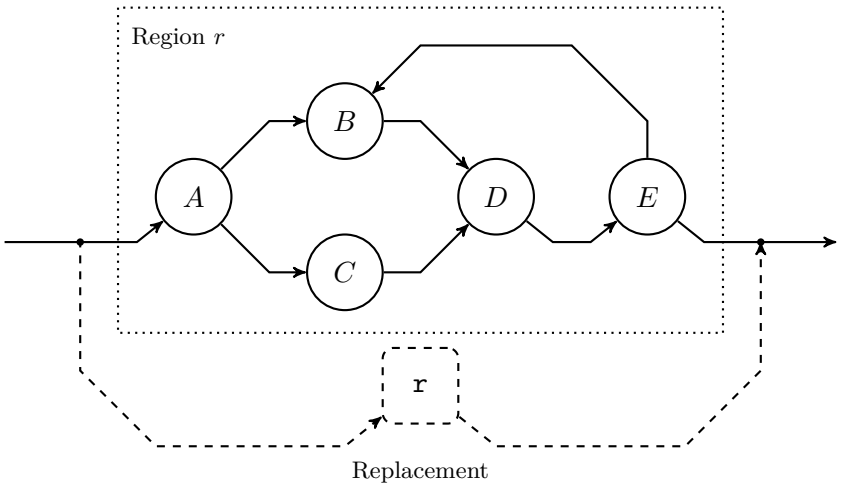


Figure 6.11: In this figure r represents a rather complex region. However, because it is still a SESE region it can be coarsened.

altering the different fractions of the different tokens would have to be buffered for the next round of tokens. Furthermore, The system would have to run down on every system in r, reproducing the assembly/disassembly. This would need to much computing power and would be too close to the computations of the original model part so that the saving of runtime while maintaining a low deviation is near to impossible. Furthermore, the ratios for the different systems are *fixed*. This description is not suitable for sampling a system with a dynamic transformation ratio. Such a system could change the ratio frequently for incoming tokens (for example, the ratio could change based on the overall situation). It is like choosing a specific gear for driving a car and not be able to change the gear unless the motor is turned off. Of course, changing the transformation ratio for a system during a simulation run would be possible. However, this would reset the state of the region (aka turn off the motor, change the gear, restart the engine).

6.6 Coarsening Arbitrarily Connected Systems

The previous chapter showed how a group of sequentially connected systems can be coarsened. State transfers and extensions for complex token processing operations where discussed. Unfortunately, not all production facilities are implemented as sequentially lined up machines. Therefore, this concept is designed to also coarsen arbitrarily connected systems. One (rather complex) example is shown in Figure 6.11. The method has some restrictions compared to the previous one as it does not support the assembly or disassembly of tokens. This is due to the arbitrary structure such a region can have and the endless possibilities to combine assembly/disassembly coming along with it.

However, the altering of tokens as it is defined in Chapter 6.5.4 is supported.

6.6.1 Sampling Groups of Arbitrarily Connected Systems

Given is a region that has an arbitrary internal connection structure. The structure can especially contain branches (and their counterpart the *joins*) in the form of several outgoing channels like the branches from A to B and C in Figure 6.11. Furthermore, the branching of the token flow can be actively controlled. A controlling mechanism can adapt the token flow along the branches based on the current situation.

Not only do we have to sample the lead time for the tokens but also we need to know which tokens are taking which route. This can be done by simply forming percentages of the token flow for the different routes. This information is needed when performing a state transfer during the switch $\mathtt{r} \rightarrow r$. Then the possible paths through the original region are reconstructed from this information. However, this system has its drawbacks. Let's assume that several locations with branches are present in the region. Then each branch location is examined independently. Conditional token flows aren't probably sampled. An example for a conditional token flow could be:

> If k moves along production line one, then it is 50% more likely that k is scrapped at the quality control than when it is produced on production line two.

Furthermore, supporting the altering of tokens gets tricky as there are usually also conditions when altering is applied. Instead of sampling all of these different components independently from each other, samples in the form of paths through r are taken. That means, for each token the path the token takes through the region is recorded. A path $W_k := (S_0, S_1, \ldots)$ of a specific token k is defined as the sequence of TPS the token visited. When k leaves the region through the exit W_k becomes a sample and is stored in a look-up table $[W]_r$. With this information the coarsening method previously used for sequential connected systems can easily be adapted for arbitrary regions. In a sequentially connected region the sequence of TPS a token visits is fixed and predetermined. Now, to coarsen an arbitrarily connected region the predetermined sequence simply can be replaced by the recorded paths.

The size of $[W]_r$ is dynamic and based on the size of the set of known paths $\mathcal{W} := \{[W]_r\}$. The assumption is, that in a situation where tokens flow through a small number of different paths only few samples are needed. However, if the tokens take a lot of different paths much more samples are needed to probably sample the distribution of the paths. This ensures that large regions with sparse usage are not oversampled.

Implementation of r

As for sequentially connected regions, r is implemented as a delaying component with a specific size. But instead of having a fixed size based on the region to coarsen (cp. Chapter 6.5.2) here the size is derived from the set of known paths. Given $[W]_r$, the size of r is defined as $\sum_S |P_S|$ with $S \in \{[W]_r\}$. This ensures that r can hold a maximum number of tokens that approximates the current situation in r.

6.6.2 Switch to the Coarsened Version ($r \rightarrow$ r)

Let $k \in r$ be a token in an arbitrarily connected region r. On transfer to r a point in time must be chosen when the processing of k is finished. For this the Equation 6.19 is utilized where a lead time related to a path is used. We simply randomly chose a path $W \in [W]_r$ from which the lead time according to (6.17) is calculated.

6.6.3 Switch Back to the Original Version (r $\rightarrow r$)

We can almost entirely rely on the switching method developed for sequential systems. For each token in $k \in$ r a path $W \in [W]_r$ is chosen and used to calculate the system (and option) to which k is transferred. The chosen path W can be seen as a sequentially connected region of the material flow path. Therefore, with a given path W simply the equations from (6.20) can be applied.

6.6.4 Summing Up

Given is a material flow model M that is based on the specifications that were developed in Chapter 6.2. During simulation the behavior of the different material flow components is sampled. The samples are stored in component specific look-up tables (Chapter 6.3). With the modified PST algorithm presented in Chapter 6.4 it is possible to partition M into a set of SESE regions. Chapter 6.5 and 6.6 both presented methods how to coarsen SESE regions. On coarsening a region r is replaced by a single, well parameterized material flow component. The parameterization is derived from the behavior samples taken during simulation.

The set of SESE regions identified with the PST algorithm are not disjunct from each other. Instead, one region may inherit several others. Based on this inheritance the regions form a hierarchy. Obviously, it is not possible to coarsen a child region of an already coarsened parent region. This circumstance introduces several coarsening options which are mutual exclusive. In the next chapter a controlling algorithm is presented that uses the behavior samples as a guidance to choose from the set of coarsening options.

6.7 Controlling the Coarsening Process

Why is a controlling mechanism needed? With the introduction of the region hierarchy in Chapter 6.4 for a given hierarchy a lot of different coarsening options exist. As pointed out in Chapter 6.4.3 even for the simple example from Figure 6.8 five different options exist: 1) Exclusively coarsen region r_1 or 2) region r_2 or 3) region r_0. 4) Coarsen both r_1 and r_2 at the same time or 5) do not coarsen anything. A trivial algorithm would chose chose an option and stick to it until the end of the simulation. However, in this chapter a controlling algorithm will be presented that evaluates and chooses regions to coarsen, based on the current model state and user defined criteria. This leads to the main question *when* and *where* and for *how long* should the coarsening method be applied?

Controlling means to specify a reference output and to transform it into an input so that the controlled system generates the desired output (cp. Chapter 2.4). In the next section the input parameters for the control function are specified. After that the controlling mechanism is presented. The mechanism is separated into the three parts *where*, *how long* and *when* which are examined independently.

6.7.1 Reference Output and Feedback

There are several parameters that influence the decision of *when*, *where* and *how long*. As discussed earlier, coarsening regions of a model will *usually* cause it to compute a different simulation output - an error in comparison to the unaltered model. This error should be kept small for result validity. On the contrary, by coarsening a larger region of the model a larger speed-up is gained. Usually the assumption that the introduced output error becomes larger, the larger the coarsened region gets holds (cp. experiment results in Chapter 8.5). That means, there is a trade-off between the speed gain and the size of the output error. A user defined parameter specifying the preferred trade-off is the *reference output*.

However, the output error is not only influenced by the size of the coarsened region. Instead, other parameters must also be taken into account. It is crucial for a valid simulation output to avoid the coarsening of bottlenecks within the material flow system (cp. Chapter 4.3). Furthermore, the whole coarsening method is based on the idea that samples can be used to guess the behavior of material flow processes (for a short amount of time). Obviously, this works better for processes that show a steady behavior. Therefore, the model state in form of the samples taken in Chapter 6.3 and a set of the current bottlenecks are provided as *feedback* parameters.

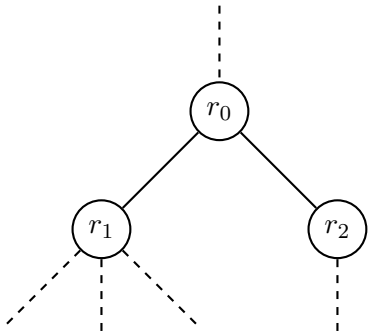


Figure 6.12: A part of a fictive $\mathcal{R}$ with three regions r_0, r_1, r_2.

6.7.2 Where?

Coarsening Efficiency

The control mechanism is built around the region hierarchy which was setup in Chapter 6.4. During simulation runtime it maintains a set of coarsened regions. The regions are chosen based on their *coarsening efficiency* rating. The coarsening efficiency is defined as the output error ϵ per speed gain μ (per token). With the reference output x the user can specify the desired trade-off between high efficiency and low output error (x is discussed in detail in the next section). This leads to the following metric for the coarsening efficiency λ for a specific region r:

$$\lambda_r = \frac{\epsilon_r}{1 + \mu_r * (1 - x)} \text{ where } x, \epsilon_r, \mu_r \in [0, 1] \tag{6.22}$$

To measure both, the speed gain and the output error, the original model, as a reference, must be available. Obviously, a complete simulation run of the original model for reference is not available. Instead, the speed gain and the output error are predicted, using two metrics that are discussed in Chapter 6.7.5 later on.

Each region in the hierarchy from Chapter 6.4 can be evaluated with (6.22) at any given point in time. A trivial controlling would be to always coarsen a specific number n of regions that have the best (lowest) coarsening efficiency. However, this would introduce with n another parameter that would have to be chosen by the user. Furthermore, n is very model specific. For one simulation model a specific n may be a good choice, for another model a bad one. For example, a specific n is a bad choice when the region hierarchy contains less regions ($|\mathcal{R}| < n$). Therefore, a different mechanism should be used that does not need additional pre-specified parameters.

The mechanism presented in the following depends on the structure of the region hierarchy $\mathcal{R}$. It tries to coarsen as much regions as possible while choosing them in such a way that the sum of their λ is as low as possible. Let $w_r := \{r, r', r'', \ldots\}$ with $r, r'r,'', \ldots \in \mathcal{R}$ be the path from region r to the root of $\mathcal{R}$. Then, at any given point in time at most one region of w_r can

be coarsened. Consider that region r_0 in the example tree from Figure 6.12 is already coarsened. That means, all nodes in that region are replaced by a single one. Obviously, it is not possible to coarsen r_1 or r_2 since their nodes are already coarsened. Consider another situation: Now, r_1 is coarsened. Since r_1 is a SESE region within r_0 it would still be possible to also coarsen r_0, but that would include the replacement component for r_1. While this is possible it introduces some complexity to the controlling mechanism: In this situation it is no longer possible to remove the coarsening of r_1 without previously removing the coarsening of r_0. Furthermore, the coarsening of r_0 is then based on samples from r_1 and not on samples of the original simulation model. This causes a much greater output error than the sole coarsening of r_0. Therefore, this *inherited* coarsening is avoided.

Given a region r and its children $c_0, c_1, c_2, \ldots$, function Φ specifies if r is chosen for coarsening:

$$\Phi(r) = \begin{cases} r & \text{if } r \text{ is a leaf,} \\ r & \text{if } \lambda_r \leq \sum_{r' \in R} \lambda_{r'}, \\ R & \text{otherwise,} \end{cases}$$

where R is defined as $R := \{\Phi(c_i) \mid i \in 0, 1, \ldots\}$.

Basically Φ tries to find a set of regions in the subtree of r for which the sum of the coarsening efficiency is lower than for r. If no such set can be found r is the best choice for the whole subtree in terms of λ. When evaluating the children with Φ the λ is simply summed up. The sum is used because the coarsened regions do not overlap and therefore the speed gain as well as the output error is assumed to accumulate.

Reference Output x in Equation 6.22

In Equation 6.22 the reference output parameter x is used to specify whether a high efficiency or a low output error is preferred. This is done by manipulating the weight of the speed gain μ with the parameter x. If x is set to zero μ is fully weighted. Thus, the higher μ or the lower ϵ - the better. If x is set to one, then μ is no longer taken into account. Then only the introduced output error ϵ counts and regions with a low output error ϵ are preferred.

The weighting mechanism is shown in Figure 6.13 for different values of ϵ and μ (depicted in the brackets as (ϵ, μ)). Imagine that λ_i is associated with r_i from Figure 6.12. It is easy to see, that λ for both r_1 and r_2 is very low. When viewed independently, r_1 and r_2 should be coarsened regardless of the specification of x. However, when combining the values of λ_1 and λ_2 it is a whole different story. When specifying a lower x ($x < 0.7$) then the efficiency for r_0 is lower (better) thus this region should be coarsened. However, with an $x > 0.7$ the output error is weighted in such that the combination of λ_1 and λ_2 has a better efficiency due to the lower output error.

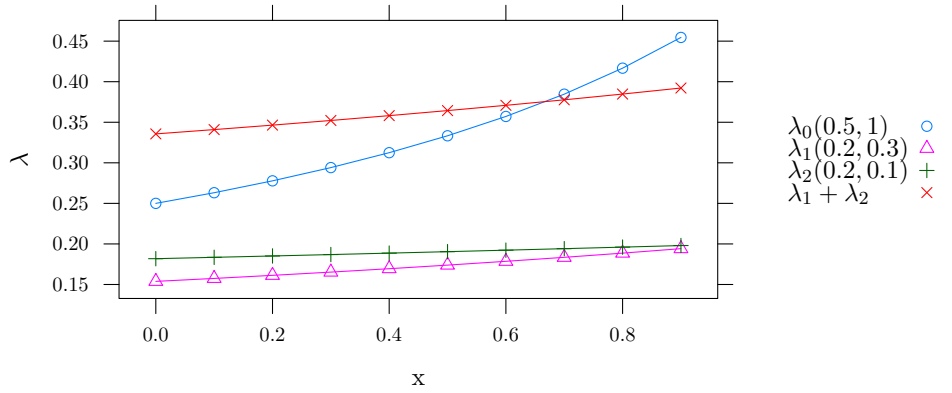


Figure 6.13: λ plotted for different values of (ϵ, μ). With an increasing x the ordering of the different λ changes due to the shifting in the weighting of μ.

Bottleneck Detection

Until now, it was assumed that any region $r \in \mathcal{R}$ can be coarsened as long as no other region on w_r (the path from r to the root of $\mathcal{R}$. See page 86) is already coarsened. Unfortunately, avoiding the coarsening of bottlenecks in a material flow system is crucial to maintain a low output error (cp. Chapter 8.3). That means if a bottleneck is present in a region $r \in \mathcal{R}$, that region or parts of that region should not be coarsened.

Most of the available bottleneck detection methods use the average of a specific ratio over a period of time to determine bottlenecks. Therefore, their reaction usually is time-delayed depending on the size of the measured time interval. Especially sudden changes like a machine breakdown can alter the location of bottlenecks within the material flow - often only for a short period of time. The more closely it is possible to follow these bottleneck changes the better the coarsening method can react and the more accurate is the simulation output. The *shifting bottleneck detection* method [RNT02] is able to identify the current *main* bottlenecks at any given point of time and can take breakdowns and other sudden changes into account. Furthermore, the method is capable of detecting secondary and tertiary bottlenecks and has been shown to work well for both complex material flow models and components with complex behavior [RNT03, WZZ05].

To constantly track the location of the bottlenecks within the simulation model a modified version of the *shifting bottleneck detection method* is used. At any given point in time, the original method identifies the component being the longest in state *active* as the main bottleneck. However, this can

lead to false detection for machines that simply have very long processing times compared to others. To reduce these false detections to a minimum, a component is considered as a main bottleneck only when it is *active* and at least one predecessor has at least one token in state ●. That implies, that the component is accumulating work and actually is a bottleneck for preceding components. Everything else of the *shifting bottleneck detection* method is left untouched. However, to quickly react to changes in the bottleneck location the time interval examined for secondary bottlenecks is chosen to be short.

Every region within the hierarchy that contains a component identified as a bottleneck is marked as *not coarsenable*. Regions containing marked regions are also marked accordingly. Obviously, this implies that the whole simulation model (the root of $\mathcal{R}$) can only be coarsened if there is no bottleneck present anywhere in the whole material flow graph.

6.7.3 How Long?

With a chosen set of regions that will be coarsened the next question is, for *how long* should they be coarsened. In this section a function $\Psi(r)$ will be defined that specifies the maximum amount of tokens that will be processed by the replacement `r`.

Predicting for how long the samples in the look-up tables reflect the current situation in a region is arguably hard. Especially, when there is only sparse data in the form a few samples available and nothing is known about the structure or type of the processes that produced these samples (for generality reasons processes weren't specified in Chapter 6.2). The goal is to derive from ϵ a value for how long the coarsening should be applied. However, there are some problems that must be overcome:

- Specifying a period of time for the length of the coarsening is not well suited. It is not possible to control (or predict) how many tokens arrive during the specified period. Therefore, the ϵ and μ cannot be controlled or predicted. Imagine a region where very few tokens arrive. It could happen that during the specified period of time no tokens arrive. That would render the coarsening obsolete and simply would cost runtime. Instead, the period how long the coarsening method is active should be expressed in numbers of processed tokens.

- ϵ is positive but can become infinite large. This makes it difficult to directly derive a value for the length for the activation of the coarsening from it.

- It should not matter how ϵ is defined. Especially, the user should not need to adapt the mapping of ϵ to *number of tokens* for a specific simulation model. This would imply that the user knows how the

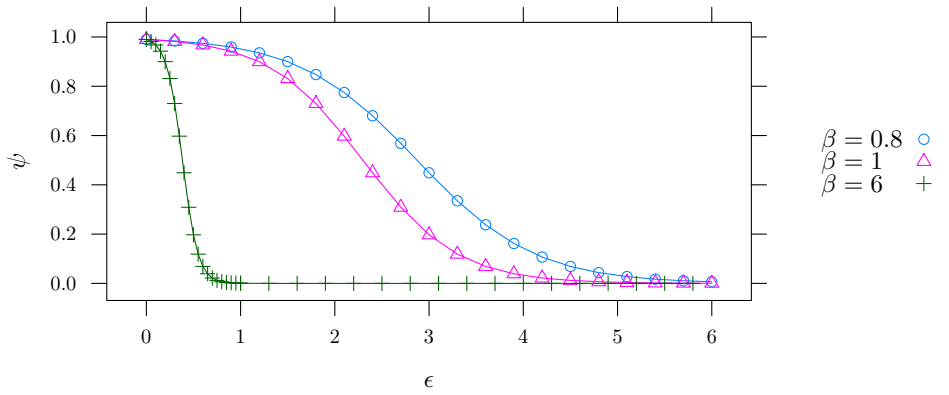


Figure 6.14: Curve of function ψ for different values of β. β can be used to adjust the ψ mapping of the ϵ into the interval $[0, 1[$.

algorithm and the error measurement works. Instead, if the user has to provide further information, it should be intuitive.

- The mapping should include the reference output x specified earlier. For example, on $x = 1$ the length of the activated coarsening should be very short to prevent much error. However, it still should save runtime. On the other had, on $x = 0$ the method should be very efficient.

- This is another mapping problem: With a (possibly) infinite large output error, when is the method efficient? How much error can be tolerated for speed? Since there is no defined boundary for the speed gain the output error could grow infinite large.

To solve these problems the output error in the domain $[0, \infty]$ will be mapped into the normed co-domain interval $[0..1[$. This is done through Equation 6.23:

$$\psi_\beta(x) = \frac{1}{2} * (1 - \tanh((x - o_\beta) * \beta)) \tag{6.23}$$

$$\text{with } x \in [0, 1] \text{ and } o_\beta = \frac{1}{\beta} * \text{atanh}(1 - 2 * 0.99).$$

Equation 6.23 uses a modified Tangens Hyperbolicus to map an arbitrary, positive value to the interval $[0, 0.99]$ (0.99 is an arbitrarily chosen but fixed value). β is a parameter that can be used to adapt the mapping to the definition of ϵ. It controls how fast ψ drops off to zero. Examples of ψ are

shown in Figure 6.14. The parameter β must not be specified by the user but instead can be adapted once to the ϵ prediction.

With the mapping $\epsilon \to \psi$ in place a mapping $\psi \to \Psi$ can be implemented. This again is tricky as the length of how long a coarsening can remain active can be infinite long. Therefore, the user has to specify an additional input parameter y that represents an upper boundary for the number of tokens. To simplify things, the user is not required to specify a certain amount of tokens that should be processed by a coarsened region before switching back to the original part. Instead, the user specifies the amount in terms of y times the amount of tokens needed for a break-even. *Parameter y*

Let's examine this further: Computing and controlling the coarsening of the regions takes runtime. Computing how tokens flow through a coarsened region usually take less runtime than in the original model. Therefore, after a certain amount of tokens a break-even is reached where the time taken for computing and controlling the coarsening method are recovered. Each token processed after that can be counted as a savings in runtime. Letting the user specify a certain amount of tokens is counter-intuitive. It is not clear to the user which amount of tokens is *much* or *little*. However, the concept of the break-even is clear and intuitive. Therefore, the user specifies with y the maximum coarsening length in terms of *break-evens*. Let ω_r be the break-even for a specific region r, then the amount of tokens processed by the coarsened region `r` usually is specified by

$$\Psi(r) = \underbrace{2 * \omega_r}_{\text{minimum}} + \underbrace{\omega_r * y * \psi(r) * (1 - x))}_{\epsilon,x,y \text{ depending}} \tag{6.24}$$

Equation 6.24 has two parts: The first part is $\omega * 2$. This part ensures, that the coarsening of a region is at least as long active to reach the break-even and to save runtime for one times the break-even. In the second part the maximum y specified by the user is multiplied with ψ (6.23) such that a low error allows an active time of y while a large error will reduce y to near zero. Furthermore, also x is taken into account. On one hand, a low x means that y should be taken into account as much as possible (as much as it is allowed by the error mapping ψ). On the other hand, a x near one means, that the focus lies on a low output error and the impact of y and ψ on Ψ is reduced to almost zero. Therefore, the parameter x has the same semantic as in the previous section. Furthermore, with a correctly computed ω, y is independently from the structure and size of the original model and its region. Therefore, with the parameter y the user can specify a maximum for the coarsening length in a intuitive way.

Measuring the Break-Even ω

The break-even is the number of tokens that has to be coarsened so that the runtime costs of the coarsening equal the runtime savings. Runtime costs

are generated by

- the control mechanism implemented in this chapter. It needs runtime for choosing the set of regions to coarsen and to observe the overall model for changing situations (see next section).
- The state transfer as discussed in the Chapters 6.5 and 6.6 cost runtime.
- The maintenance of the look-up tables constantly costs runtime. Every event produced by the TPS during runtime has to be processed to generate samples for the look-up tables.

On the other side, runtime is only saved by omitting events due to the omitting of model state changes. This includes the processing and movement of tokens within a TPS as well as between two different TPS. Because of the diversity of the costs, measuring the break-even is a difficult task.

However, in a single preprocessing step the different costs can be measured using artificial simulation models like the Q and F models described in Chapter 8.1. These can be generated in different sizes such that it is possible to implement a regression model for the costs. This model then can be used to compute a break-even based on the current situation.

Identification of Situation Changes

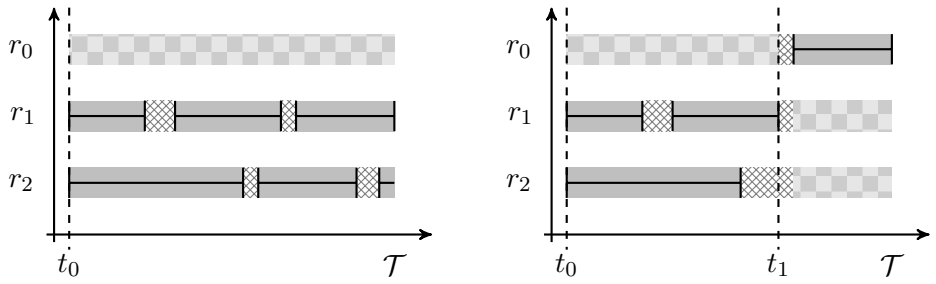
Up to now the length of an active coarsening was computed from the output error ϵ and two user specified parameters x and y. However, there are some situations when the validity of the currently used samples can be doubted. The samples in the look-up tables reflect a certain situation when they were taken. This includes current processing as well as waiting times. The former can be controlled by the TPS themselves. The latter reflects a certain situation defined by the incoming and outgoing stream of tokens. It specifically reflects a certain ratio between these two streams. If the ratio changes, because one of the two streams (or both) change then the waiting times will also change.

Given is an arbitrary SESE region. Suppose, that the input stream of tokens to that region increases while the outflow of tokens remains constant. Then the current *work in process* will increase. Obviously, the *work in process* will decrease when the outgoing stream increases under a constant input stream.

Therefore, a increasing or decreasing *work in process* can be used as an indication of situational changes within the model and that a resampling should be done.

6.7.4 When?

Of course, there are some restrictions that must be respected before coarsening can be applied. First of all, enough samples must be available. The look-up



(a) The problem: Because r_1 and r_2 are coarsened alternately, there is no opportunity to coarsen r_0 which remains inactive .

(b) The solution is to resample the whole simulation model at specific times (t_1). Based on this new data re-choose the regions to coarsen (here r_0).

Figure 6.15: Exemplified coarsening , resampling and inactive times of the three regions from the example in Chapter 6.4.3.

tables must be completely filled to make an informed decision and to do a high quality state transfer when switching. Also identified bottlenecks within a region restrict the applicability of the coarsening process. In sequentially connected regions, the bottlenecks can simply be spared from coarsening. The bottlenecks cleanly partition a sequentially connected region into smaller sequentially connected SESE regions. They can be coarsened in the usual way. However, in arbitrarily connected regions this is not possible since the bottlenecks do not cleanly separate the region into smaller SESE regions. Therefore, a bottleneck blocks the coarsening of arbitrarily connected regions.

When coarsening several regions at the same time another problem arises: Because each region will be coarsened for its own individual amount of tokens and due to a nonuniform distributed token flow it is save to assume that each region will be coarsened for a different amount of time. These very individual coarsening times will usually overlap. There is no point in time where the whole system can be resampled to re-choose the set of regions to coarsen.

A small example will explain this problem and its solution in detail. Given is the simple region hierarchy from Chapter 6.4.3. As depicted in Figure 6.15a at time t_0 it was decided to coarsen r_1 and r_2. As explained earlier both regions have very different time periods where they are coarsened. Of course, at some point in time each region is returned to its original component structure and is being resampled . This individual resampling of r_1 and r_2 does not overlap due to the different coarsening times (cp. Figure 6.15a). Since both, r_1 and r_2 are a part of r_0 (cp. Figure 6.8) there is no point in time where samples could be taken for r_0. Furthermore, r_0 is completely blocked at any point in time where at least one of its children is coarsened. Thus, there is no possibility to reevaluate the option to coarsen r_0 instead of r_1 and r_2. To solve this problem, a time period is introduced where the

whole original simulation model is executed and resampled (t_1 in Figure 6.15b). Based on these new samples all regions are reevaluated and a new set of regions for coarsening can be chosen. This set is *locked* until the next point in time where the whole simulation model is reevaluated. Based on Equation 6.24 for each region a specific amount of tokens is calculated, that represents the maximum of tokens that will flow through the coarsened region. When the last of the regions identified by Φ has finished its assigned amount of tokens the original simulation model is reevaluated and a new set of regions for coarsening is chosen. If a region finishes early, it is most likely a region with a high throughput of tokens. Then this region is resampled and coarsened again. This ensures that these regions are coarsened over the whole time. This is shown for r_1 in Figure 6.15b.

Restrictions like bottlenecks are still tracked and the coarsening of a region might be canceled due to a shifting bottleneck or a change in the overall model state. Such a cancelation counts the same as if the assigned amount of tokens that should flow through the coarsened region has been reached.

6.7.5 Measuring Speed Gain and Output Error

In Chapter 2.3 the difference ϵ between two simulation models M and M' was specified as the difference in the decisions being made from the simulation results from the two models. However, as stated in Chapter 3 decisions (or the difference between them) cannot be measured. Therefore, ϵ was redefined as being the difference in one of three measurable ratios, namely: *lead time*, *throughput* and *work in process*. Now, Equation 6.22 uses the output error ϵ and the speed gain μ. Both ratios are assumed to be measurable and are measured in relation to the original simulation model. That means, for a given simulation model M ϵ specifies the difference in the simulation results for a variant M' where the coarsening is active. The same goes for the speed gain. The speed gain *SpeedGain* specifies the amount of time that is saved when activating the coarsening concept in relation to the simulation of the original model M. To measure both ϵ and μ the original model must be simulated for reference. However, when simulating M anyway then the coarsening concept is needless. Therefore, the output error (or difference) ϵ and the speed gain μ have to be predicted.

Predicting μ

The speed gain can be approximated by the number of omitted events when processing a token (*Runtime Complexity* [SY93]). It can safely be assumed that all token processing systems need a similar, constant number of events to process a token. Therefore, the speed gain depends on the (average) length of the distance a token covers while moving through a region r. For a

sequentially connected region this is its size $\|r\|$. For arbitrarily connected regions its the average length of the ways contained in $[W]$. Furthermore, the speed gain depends on the amount of tokens that runs through a coarsened region. Each token (in average) contributes the same to the speed gain. Therefore, also Ψ is taken into account.

$$\hat{\mu}_r = \Psi * \begin{cases} \|r\| & \text{if } r \text{ is sequential and} \\ \frac{1}{\|[W]\|} \sum_{w \in [W]_r} \|w\| & \text{otherwise.} \end{cases}$$

For the predicted speed gain when activating coarsening the symbol $\hat{\mu}$ is used.

Predicting ϵ

The output error is hard to measure. Especially, because it depends on the differences of the decisions being made. Instead, the variance of the values in the look-up tables is used as an approximation. The variance is an absolute value, making it hard to interpret without a context. Suppose, you want to meet a friend at a shop. Obviously, there is a difference between the two statements *"I'll reach the shop in ten to thirty minutes."* and *"I'll reach the shop in one day plus or minus 10 minutes."*. Despite the variance being the same in both examples (20 minutes), in the first example one would say, that you are very unpredictable while in the second example you would seem very predictable. That is because of the context (here the overall time frame) is very different. Therefore, the (standard) deviation related to the mean of the observed values is used as an approximation $\hat{\epsilon}$ for the size of the output error.

$$\hat{\epsilon}_r = \frac{\sigma([lt]_r)}{\varnothing([lt]_r)}$$

where σ depicts the standard deviation and $\varnothing$ the arithmetic mean of the samples.

6.8 Conclusion

In this chapter a coarsening method has been presented that is designed to be applicable to (almost) arbitrary material flow models. This is accomplished because of two things: First of all, the material flow system specification only specifies locations with places for tokens and actions on them. Processes controlling and executing these actions have been left undefined intentionally. Therefore, the coarsening method is compatible with almost any material flow component. Furthermore, this coarsening concept is capable of coarsening arbitrarily connected single-entry-single-exit structures. It does not rely on certain graph structures to be present. However, to support the altering

and assembly and disassembly of tokens specifications for token processing systems where introduced. The three aforementioned processes have to be specified as fixed functions such that the coarsening method can utilize them when it is active.

Furthermore, the coarsening method does not need a computational and time intensive preprocessing. Instead its preprocessing takes only milliseconds for reasonable sized models (see Chapter 8.7.1). Thus it can be applied during runtime at will and it is fast enough to adapt itself to dynamic changes of the material flow structure. This allows the usage of the coarsening method even during model construction.

The controlling of the coarsening method is designed to be understandable and user-friendly. Model and simulation system dependent parameters have been avoided and the user does not have to specify parameters that are only understandable with deep knowledge of the coarsening and simulation method. The user simply provides two parameters x and y. The former specifies a trade-off between efficiency and error avoidance. The latter defines a maximum amount of tokens when coarsening regions - in terms of break-evens. All other (internal) parameter are determined by the coarsening method itself.

With the utilization of the *shifting bottleneck* detection method the coarsening method is able to adapt itself to model dynamics (model state changes). Especially sudden break downs of machines can be tracked with this method. For a quality simulation output the bottlenecks are respected when choosing the material flow regions to coarsen.

Chapter 7

Implementation

The concept for a coarsening method for material flow models that was presented in the previous chapter, has been implemented into the d^{3}fact simulation platform for evaluation. The design and implementation steps that were necessary for the concept implementation are outlined in this chapter.

The usual approach to create a specific material flow model is to connect black boxes, representing the processes carried out on the tokens. Unfortunately, this approach allows no state tracking for components. Therefore, in Chapter 6.2 a more transparent material flow specification has been outlined. The implementation of this specification is presented in the next section. On top of this implementation the different subsystems of the coarsening concept have been realized. Their implementation is discussed afterwards.

7.1 Material Flow System Implementation

The specifications from Chapter 6.2 have been implemented as a new component library to replace the previous approach outlined in Chapter 4.5.4. The specification defines two main components and their dynamics: *Token Processing Systems* (TPS) and *Channels*.

7.1.1 Token Processing System Implementation

A TPS is essentially specified as a local storage for tokens which allows specific state changes from external. The storage itself is a set of identifiable places, where each place can be in one of four states: {occupied, not occupied} $\times$\{ accessible, inaccessible }. When occupied, the place stores a specific token $k \in \mathcal{K}$. The implementation of a specific material flow component

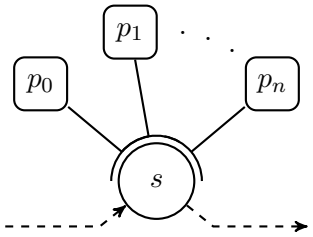


Figure 7.1: A storage location s, associated with a set of processes $p_0, p_1, \ldots, p_n$ and an incoming and an outgoing channel (indicated as dashed lines).

like a conveyor therefore consists of two parts: The storage location and a controlling process that imitates a specific behavior, e.g. that of a conveyor. That means, the process moves the tokens within the location as if they were lying on a conveyor belt. Processes can also represent moving entities like fork lifts. Then the process maintains a position within a scene, e.g. a warehouse. The implementation explicitly separates the token storage from the processes. This allows some very interesting setups.

The implementation of the specification allows the association of several processes to one location and also the association of one process with several locations. This maps to situations, where e.g. several workers work on the same set of tokens. Another useful application are *automated storage and retrieval systems* (AS/RS) where e.g. several of these access the same rack.

7.1.2 Channel Implementation

The channel implementation works very much the same as specified in Equations 6.10 and 6.11. Each channel is implemented as simulation object.

After all the processes at a TPS have been informed about state changes in the storage location the channels are informed per simulation event. Based on the current state of the location they compute their *enabled* state. If a channel is *enabled* it immediately transfers tokens from the source to the destination. This usually renders all subsequent channel disabled.

7.1.3 Implementation Details

Let a location s have n associated processes $p_0, \ldots, p_n$ as depicted in Figure 7.1. Now an arbitrary process $p_s \in \{p_0, \ldots, p_n\}$ makes changes to the location. This creates an update u_s containing all information about the changes, i.e. added and removed entities, now accessible or inaccessible places. Unfortunately u_s starts an update cascade. That means, another process p_k responds with its own update u_k to the update u_s. Now u_k again causes another process p_r to respond, and so on. A trivial update mechanism would inform each of the n processes of every update. This can lead to

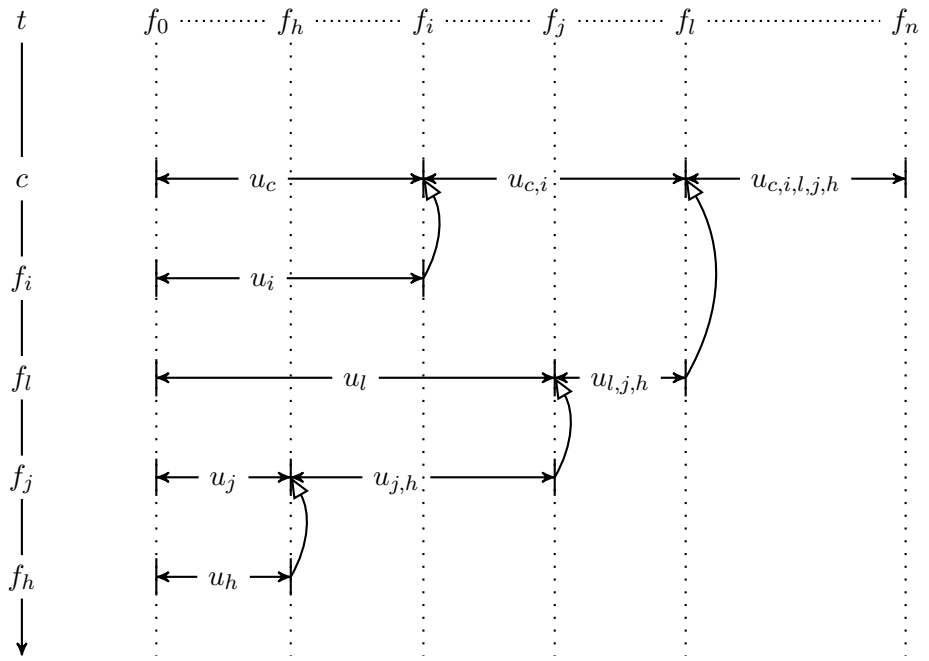


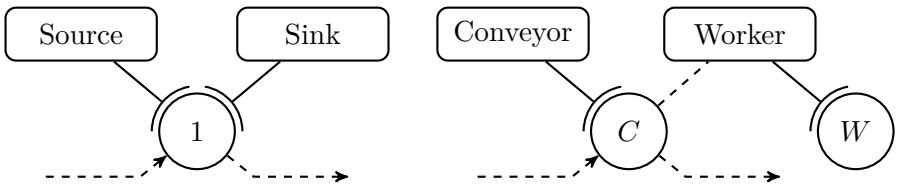
Figure 7.2: Generic example showing how our update method works. Horizontally all processes are displayed while the time line is plotted vertically.

an efficiency problem. For example one process occasionally creates new entities while another process destroys them to replace them with completely new entities. Every associated process now gets both updates, even if the entities created in the first place do not last long or have been destroyed already. Carrying out each update can lead to a huge overhead when updates contain oppositional or obsolete information. Also, processes would have to be capable of determining the obsolete information in the update or the differences between the last known and the current state of the location. This would make the implementation of new processes more difficult, especially for new users.

Instead of informing all processes about all updates, we accumulate updates. This reduces the times a process is informed about updates to a minimum and the updates do not contain stale information. The update mechanism we came up with can be found in Listing B.2. It resides in the location implementation. Changes made by a process to the location trigger the `UPDATE()` method. Here the parameter `p` is the process initiating the changes and `u` is the update initiated by `p`.

We inform the processes in the order they were added. If one process happens to start a new update during a running update we suspend the current update and start from the beginning (with the new update).

Figure 7.2 shows an exemplified update cascade. Horizontal lines indi-



(a) Minimal Example.

(b) The Worker and the Conveyor Example.

Figure 7.3: Two examples showing the usage of the described material flow implementation.

cate which process is informed about which update at which point in time. Horizontally the processes $p_0, \ldots, p_n$ are shown. The time line is displayed vertically. A back pointer indicates the accumulation of two updates. Basically the procedure starts with an update u_s. Now we consider p_k to be the next process responding to the changes made by u_s. That means, that u_s is applied to all processes before process p_k. Upon informing p_k about u_s it triggers a new update u_k. As stated before we now start from the beginning, informing all processes before p_k (cp. Listing B.2, Lines 5-12). After doing so, all processes $p < p_k$ are informed about the updates u_s and u_k. Before informing the processes $p > p_k$ we merge both updates into the new update u_{s+k} eliminating all oppositional information (Line 20, Listing B.2). As depicted in Figure 7.2 this update algorithm can also handle recursively triggered updates.

Minimal Example

The most simple non-blocking material flow system one can think of is a storage object with a *source* and a *sink* process (see Figure 7.3a). Where *source* and *sink* work as one would expect.

In this example the *source* occasionally creates tokens and places them in s. s does have a capacity of one. After placing a new token in s the *source* has to wait until the token is removed from s before it may generate a new token. Through the update mechanism described earlier the *sink* is informed about the state change in s. Now the *sink* can access s and remove the token. This clears the place for the *source* to add a new token. No other systems, processes or objects are needed. While this example is very minimalistic, it shows how the separation works and how easy it is to setup a material flow model.

The Conveyor and Worker Example

A more complex scenario would be one, where a worker walks by a conveyor. During his walk by, the worker sees a processed part on the conveyor that

is defect. Now the worker can interact with the conveyor by removing that specific part and disposing it. In this scenario, like the worker the conveyor is represented by a storage with an associated process (cp. Figure 7.3b). During his walk by, the worker process is dynamically added to the conveyor storage. This gives him the access to the parts moving along the conveyor belt. Now the worker can remove a defective part from the conveyor storage and place it in its own storage. Then the worker can go to a sink in the model and let the sink dispose the defective part.

7.2 Integrating the Token Sampling

We will see in this chapter, that, due to the explicit separation of the token storage from the processes, the integration of the *Token Sampling* method can be achieved easily. The method itself is separated into a setup and an online phase where the simulated system is constantly monitored (cp. Chapter 6.1). The setup is simply triggered by a simulation event at runtime. That ensures data consistency as the setup is done as part of the simulation itself. During the setup the sampling of the TPS is initiated as well as single entry single exit regions are identified.

The controlling process will be continuously be triggered by events indicating state changes. During the setup callbacks are registered to receive these events.

7.2.1 Token State Sampling

At first, the sampling of the token states within model components is initiated. The sampling is implemented as a process that can be added to a location. This allows the addition of such a process to every material flow component in the model. Furthermore, due to the container-based representation of simulation object in d^{3}fact (cp. Chapter 4.5.3) upgrading the model components is very easy.

Given an arbitrary material flow component with a storage location and a set of processes as depicted in Figure 7.1 then the sampling process is added as a new process p_{n+1}. This ensures that the process is informed about every location update and especially is informed before other simulation objects like channels. This ensures data consistency. The sampling process fills the look-up tables [⊗] and [●] with data from tokens moving through the storage location. The look-up tables are simply added as new properties to the simulation object. Due to the nature of the update method described in Listing B.2 this implementation of the sampling does not produce any additional events while running.

7.2.2 Identifying Groups of Systems

Having setup the sampling of the material flow components now *single entry single exit* (SESE) regions within the material flow graph have to be identified. For this a modified version of the *Program Structure Tree* algorithm by Johnson et al. [JPP94] is used (cp. Chapter 6.4). The algorithm constructs a tree where each node represents a SESE region in the model.

Since the simulation object specification of d^{3}fact supports hierarchies in general, the nodes of the PST are implemented as simulation objects and are added to the simulation model. Based on the type of the region (sequentially or arbitrary connected) specific simulation objects are created and integrated into the tree. The implementation of these objects is discussed in the next chapters.

The dynamic update of the tree structure according to changes made to the material flow is handled through a *listener* concept. Listener are simple objects that listen for certain state changes of simulation objects. If such a state change is triggered the listeners are informed about that. Especially the simulation model supports the listener concept for the addition and removal of objects.

7.2.3 Coarsening of Sequentially Connected Regions

Each SESE region in the material flow graph is represented as a node in a tree. The representation nodes are constructed in such a way that they serves as the basis for the controlled coarsening. For a sequential connected region r such a node stores the following properties:

- A reference to the entry object of r.
- A reference to the exit object of r.
- A reference to the replacement `r`.
- A tracker and a look-up table for the lead time $[lt]_r$ of tokens through r. These values are needed for the state transfer. Especially in the Equations (6.20) and (6.19).
- A tracker with a database (hash map) for the entry time of a token. This is also needed during a state transfer. $\check{e}_0$ in Equation 6.20.
- The state transfer logic for both directions $r \leftrightarrow \mathtt{r}$.
- A component that calculates λ (Equation 6.22).
- A set of conditions (see Chapter 7.2.5).

When a state transfer is triggered (in either direction) then changes to the model state are applied. For example, tokens are removed from their current location and put into a new one. This triggers simulation events indicating these changes. However, if the value tracker such as the one for the lead time of tokens does process these events the look-up tables may contain false values. Therefore, these objects must be informed before and after a switch so that they ignore events occurring in-between. To solve this problem an interface called *SwitchControlled* was introduced. Every property implementing this interface is informed about the state of a switch so that it can react accordingly.

State Transfer $r \rightarrow$ `r` This state transfer simply runs through all TPS located in r and transfers each token into the replacement. For each token an individual delay time Δt is computed, based on Equation 6.19. Listing B.3 shows the pseudo code for this process.

On each call of `PUT()` a new event is created broadcasting the state change of the location of `r`. However, due to the merging capabilities of the update method from Listing B.2 it is possible to merge all these single updates into one so that also only one event is triggered.

State Transfer `r` $\rightarrow r$ The state transfer back to the original components is much trickier. Equation 6.20 defines the optimum in terms of error reduction. Unfortunately, the equations miss the fact that in most material flow systems the capacity of P_{S_i} is limited. Therefore, the situation may arise where a token cannot be put into the optimal place.

The Listing B.4 shows the current implementation for this state transfer. Instead of blindly computing the optimum for each token the algorithm tries to put each token as close as possible to its optimum. For this the algorithm runs backwards through all TPS and fills them with tokens according to (6.20). Now the situation arises that more tokens should be put into a specific TPS as it has capacity, then the algorithm wraps the TPS up and adds the left over tokens to the next TPS. As before, the calls of the `PUT()` method can be merged into one update for each TPS and option. Therefore, at maximum two times the number of TPS in r events are generated.

7.2.4 Coarsening of Arbitrarily Connected Regions

The node representation of an arbitrarily connected region r has the same properties as a node for sequentially connected regions (cp. previous chapter). As discussed in Chapter 6.6 the state transfer approach relies on paths that tokens have taken through the region. Therefore, the node has a path tracker. The tracker listens on the channels of the region for token movement to construct for each token its path through the region. The path consists of an ordered list of TPS.

The state transfer $r \rightarrow \texttt{r}$ is done in the same way as for sequentially connected regions (Listing B.3). The algorithm performing the state transfer $\texttt{r} \rightarrow r$ is specified in Listing B.6. It fully utilizes the transfer routine `TRANSFER_TO_ORIGINAL()` already used in the state transfer for the sequentially connected region. Since the routine needs a tuple of sequentially connected TPS this algorithm uses the paths tracked during the observation of r. The paths are each served as much tokens as specified by the number of their occurrence in the look-up table $[W]_r$. If at the end there are still tokens left over, random paths (with some free places left over) are chosen for transfer.

In conclusion the handling of arbitrarily connected regions can be broken down to observed token paths and then the routines already used for sequentially connected regions can be utilized.

7.2.5 Controlling the Coarsening Process

In Chapter 6.7 the controlling mechanism is mainly described through equations. Furthermore, restrictions where described when coarsening cannot be applied or when it should be deactivated. In this section the overall integration of the controlling process is presented.

The integration of the controlling is based on simulation events. One such event indicates a change of the input parameters of the controlling process or a change of the model state. If an event occurs the current coarsening setup has to be verified if it still complies to the equations and restrictions defined earlier. If not, the setup has to be adapted according to the new situation. Using simulation events to trigger the setup verification routine has one benefit: The relatively computing intensive routine is only triggered if needed.

Conditions

The implementation of the requirements is done through *Conditions*. A condition describes whether a region currently meeds a requirement or not. For each region a set of conditions is maintained. Only when all requirements are met the coarsening is activated for a specific region. For example, the requirement that enough samples have been gathered is implemented as a check if the look-up tables are completely filled. The check is triggered every time a new entry is added to the look-up table. Because most of the conditions are simple checks the processing of model state changes is very fast. The overall, computing intensive setup validation routine is only triggered if a condition changes its state. The conditions are integrated into the nodes of the region hierarchy tree. On the contrary, there is a set of conditions that checks whether all requirements are still met during an active coarsening. That means, if at least one condition is no longer met, a switch back to the original model section is triggered. The conditions have the great benefit, that different requirements can be implemented independently from each other.

Chapter 8

Validation

In the following the methods that were presented in the chapter *Conceptual Design* will be evaluated and validated. At first each part of the concept is for itself evaluated using purpose-build models, which are described in the next chapter. From the gained results usage instructions and parameterizations for the coarsening concept are derived. This knowledge is then used to evaluate the concept as a whole. This is done for a material flow model described by Huber and Dangelmaier [HD09].

8.1 Purpose-Build Models

The presented concept consists of several parts. Namely, the model partition, the control metric, the bottleneck tracking and the coarsening concept with the state transfer. Each of these components depends on a set of different parameters and is affected in its performance in a specific way when these parameters are changed. To evaluate the impact of the different parameters, the behavior of each part will be examined solely using purpose-build models. These models are as simple as possible which makes them predictable in their behavior. Each of these models can be parametrized to represent very specific situations.

8.1.1 Model Q

The first model represents a sequentially connected SESE region (cp. Figure 8.1). It has a specific size with n pairs of buffers and delays (depicted as B_i and D_i). The buffers have a fixed size of fifteen places and the delays a fixed size of one. The rather large size of the buffers allows the observation and examination of the impact of bottlenecks. But then, buffers

Figure 8.1: Sequentially connected generic material flow model Q.

of this size can be filled easily, such that they block preceding components. Infinite or near infinite sized buffers are uninteresting for a validation as they decouple machines which minimizes blocking effects. Furthermore, there exist analytical methods like *Queueing Theory* to examine systems with these kind of buffers (cp. Chapter 4.1.6). Two normal distributions $\mathcal{N}_1, \mathcal{N}_2$ have to be specified. These distributions are used to parametrize the inter-arrival times of the source and the processing times of the delays and machines. A parameter i determines which distribution is applied to which component:

$i = 0$ In this case the source is parametrized with $\mathcal{N}_2$ and all delays with $\mathcal{N}_1$.

$i > 0$ The source is always parameterized with $\mathcal{N}_1$, whereas the delays are parameterized according the following equation:

$$\mathcal{N}_{D_k} = \begin{cases} \mathcal{N}_1 & \text{if } k \bmod i \neq 0, \\ \mathcal{N}_2 & \text{otherwise.} \end{cases} \tag{8.1}$$

where $\mathcal{N}_{D_k}$ represents the parameterization for the k-th delay.

With these parameterization rules different bottleneck situations can be constructed. For example, given are two distributions $\mathcal{N}_1, \mathcal{N}_2$ where the mean $\varnothing$ of the first one is significant smaller than the one of the second distribution ($\varnothing(\mathcal{N}_1) << \varnothing(\mathcal{N}_2)$). If parameter i is set to n, only the last delay D_n in the sequence is parameterized with $\mathcal{N}_2$. That means, the last delay D_n is the bottleneck. In general a model Q can be specified as a 4-tuple $Q(n, i, \mathcal{N}_1, \mathcal{N}_2)$.

8.1.2 Model F

The second purpose-build model F can be used to study the impact of branches within the material flow (cp. Figure 8.2). Basically, this model represents one large switch with n different branches. The same parameterization rules described for model Q apply for this purpose-build model. Like model Q model F can be specified as a 4-tuple $F(n, i, \mathcal{N}_1, \mathcal{N}_2)$.

8.2 Measurement and Evaluation Methods

All of the following experiments were implemented with the d^{3}factsimulation software. All experiments (especially the runtime measurements) were exe-

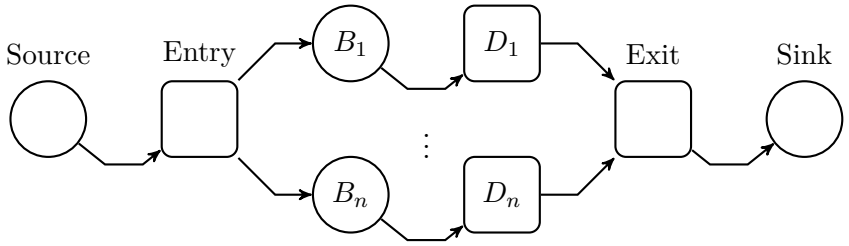


Figure 8.2: Arbitrarily connected generic material flow model F.

cuted on a laptop with a 2.4 GHz Dual Core 2 Processor and 8 GB Ram. Each experiment was run exactly 50 times and if not stated otherwise the mean value of these 50 data sets was used for evaluation.

As stated in Chapter 5 one problem that has to be solved is the quantification of the difference in the simulation output when utilizing the coarsening method. In Chapter 2.3 this difference was defined as the difference in the decisions that are made, based on the simulation results of the original model M and the coarsened variant M'. However, decisions are hard (to impossible) to quantify. Instead, the difference was redefined as being the difference between $Y(M)$ and $Y(M')$ (cp. Equation 2.4). $Y(.)$ is very specific to the current object of investigation. That means, while the simulation run is the same $Y(.)$ can change as a different ratio is measured. In practice usually the *lead time* of the tokens or the *cycle time* of the model is used as $Y(.)$ [BT00, Ros07, Ros99, JLGL99, HL99, Mer05, JFM05]. However, focusing on a single ratio can lead to the false assumption that the proposed method still works (well) for other measurements. Huber [Hub09] solves this problem by creating a single ratio from three standard ratios, namely *lead time*, *throughput* and *work in process*. However, the aggregation of these three ratios into a single metric can lead to the same problem as focusing on a single ratio: Large differences in one ratio can be cleared by small differences in others. For a more impartial view on the performance of this coarsening concept the three ratios specified by Huber are measured and compared independently. Therefore, in the following two to three different values for the same experiment are shown. The ratio names are shortened as follows: *lead time* (LT), *throughput* (TP) and *work in process* (WIP).

Ratio Measurement

The ratios are always measured for the whole simulation model. In the following, let $[t_a, t_b]$ be a period of simulation time. During this time, c tokens are created at the source(s) and a set D of tokens is destroyed at the sink(s). Given a destroyed token $k \in D$, it has a creation time t_k^c and was destroyed at time t_k^d. Then the average *lead time* per token for the time period $[t_a, t_b]$ is measured as

$$lt := \frac{\sum_{k \in D} t_k^d - t_k^c}{\|D\|}. \tag{8.2}$$

The average *throughput* per point in time for the period $[t_a, t_b]$ is specified as

$$tp := \frac{\|D\|}{t_b - t_a}.$$

Given Equation 6.1 and 6.4 then the average *work in process* per point in time is specified as

$$wip := \frac{\int_{t_a}^{t_b} \sum_{S \in \mathcal{S}_\mathcal{K}} \parallel P_S\|_t \mathrm{dt}}{t_b - t_a},$$

where $\parallel P_S\|_t$ specifies the amount of tokens that is present in the place set P_S of the token processing system $S \in \mathcal{S}_\mathcal{K}$ at a specific point in time t. $\parallel P_S\|$ is defined as

$$\parallel P_S\| = \sum_{p \in P_S} \begin{cases} 1 & \text{if } q(p) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

$q(p)$ is a function that describes which place $p \in P_S$ of a location S contains which token - if any (also see page 62).

Relative Output Error

In the following charts, usually on one of the axis the *relative output error* is plotted. The relative error for a coarsened simulation model is the relative difference of one of the ratios in relation to the original model. Let's explore that in more detail: Given is a simulation model M and a coarsened variant M'. During simulation the three ratios LT, TP and WIP are measured in time intervals of fixed size. Exemplified, let lt_M be the average lead time for M and $lt_{M'}$ the one for M' for a specific time interval (according to Equation 8.2). Then the relative output error $\epsilon_{(\text{in }\%)}$ for the coarsened variant in reference to the original model M for the lead time is specified as

$$\epsilon_{(\text{in }\%)} := \frac{\|lt_{M'} - lt_M\|}{lt_M}$$

8.3 Do not Coarsen the Bottleneck

Johnson et. al. [JFM05] point out that the behavior of the coarsened variants correlate well with the original model when the machine that is the bottleneck, was preserved and not coarsened. However, in this concept the replacement component is parameterized with samples from the original model. If there is a bottleneck located in the coarsened part of the model,

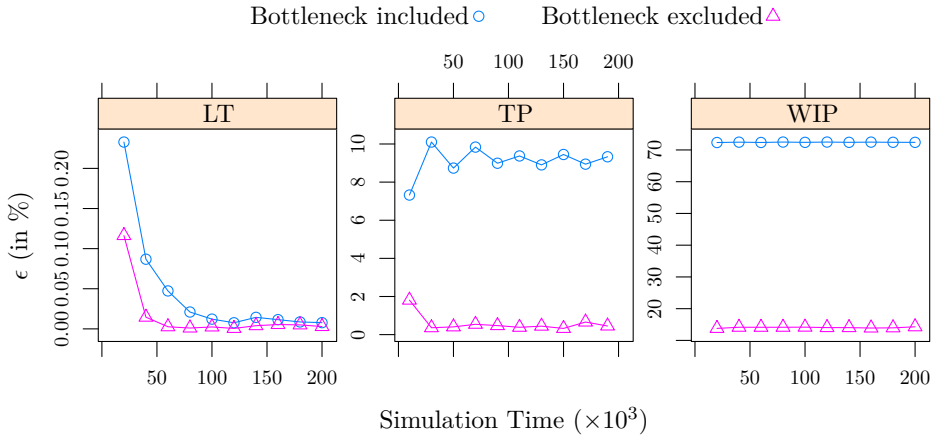


Figure 8.3: This diagrams shows the relative error ϵ (lower is better) for the three ratios when including and excluding the bottleneck.

the samples should appropriately emulate its behavior. Furthermore, due to the frequent resampling (and the state transfer) taking place, the method should be able to keep track of the behavior of the original model.

To point out the impact of the bottleneck on the overall model behavior a rather simple model setup is used: $Q(11, 6, \mathcal{N}(10, 0.5), \mathcal{N}(11, 0.5))$. The bottleneck D_6 is located in the very middle of the model. There are five delays and buffers located in front of it and five delays and buffers behind it. Intentionally, the difference between the bottleneck and the rest of the model is very subtle. Due to a variance of $\sigma^2 = 0.5$ there are almost no variances in the processing times. Therefore, the behavior of this simple model should be very predictable, which makes it easy to coarsen it with a very small error in relation to the original model. Two different coarsening setups were simulated and were compared to the measured ratios of the original simulation model. In the first setup the whole region from B_1 to D_{11} was coarsened. In the second setup D_6 was excluded from coarsening. Instead, the two regions from B_1 to B_6 and from B_7 to D_{11} where coarsened. In Figure 8.3 the relative difference $\epsilon_{(\text{in }\%)}$ between the two coarsening setups and the original model is plotted against the simulation time. One dot in the diagram depicts the relative output error for the time interval between the current point in time and the point in time of the previous dot.

The concept works very well for the *lead time*, well for the *throughput* but shows a rather large difference of 70% for the *work in process*. The relative error for the lead time ratio decreases over time and almost reaches zero. This happens because, over time the bottleneck kicks in and the tokens pile up before it. Therefore, the lead time for the tokens increases and fluctuations

become marginal. The throughput works very well when the bottleneck is excluded (ϵ near zero percent) and still works quite well when including the bottleneck with an ϵ below ten percent. However, when measuring the work in process, the bottleneck should *always* be excluded, otherwise for this ratio the error reaches 70% (at least for this configuration). This happens most likely because the replacement component `r` processes the tokens in parallel omitting the bottleneck as the dominating component that processes the tokens in serial. The difference for the work in process emphasizes even more when recalling that the model is very easy to predict in its behavior due to its very low variance.

In conclusion, when the object of investigation is the lead time of the tokens identifying and omitting the bottleneck from coarsening is of no interest as the difference to the original model is very low. However, for other ratios like the throughput and work in process, identifying and omitting the bottleneck from coarsening leads to significant smaller errors. Most likely, this is also true for other ratios and measurements not covered in this experiment.

8.4 Error Size Dependency

To implement an efficient coarsening method it must be understood which parameters affect the size of the error that is introduced when applying the coarsening method. The error size is most likely affected by the size of the coarsened group and the resampling rate. Of course, if the coarsened model part does show very predictable behavior there will be (almost) no error at all. Therefore, in this experiment a dynamic version of model Q was tested.

During simulation at a fixed interval rate every machine that was parameterized with distribution $\mathcal{N}_2$ was switched to $\mathcal{N}_1$ for a short period of time, thus removing the bottleneck. In this dynamic environment the overall model size as well as the bottleneck location (parameter i) and the resampling rate (parameter y from Page 91) were increased and the difference between the original and the coarsened model was recorded. Figure 8.4 shows the results for the two ratios *throughput* and *work in process* for two exemplified cases ($i = 0$ and $i = 10$). *Lead time* has been omitted as it shows a very low error of less than four percent in all cases.

The error size is noticeably higher in the $i = 0$ case for both ratios. In this case, the source is the bottleneck, as it is the only component that is parameterized with $\mathcal{N}_2$. Then the processing times of the machines (and their variadic behavior) have a large impact on the overall token processing. Because of this impact token processing is hard to imitate with samples. While the error size depends on both, n and ϵ in the $i = 0$ case, for small y the size of n does not matter. That means, high resampling rates of the components allow the coarsening of large regions.

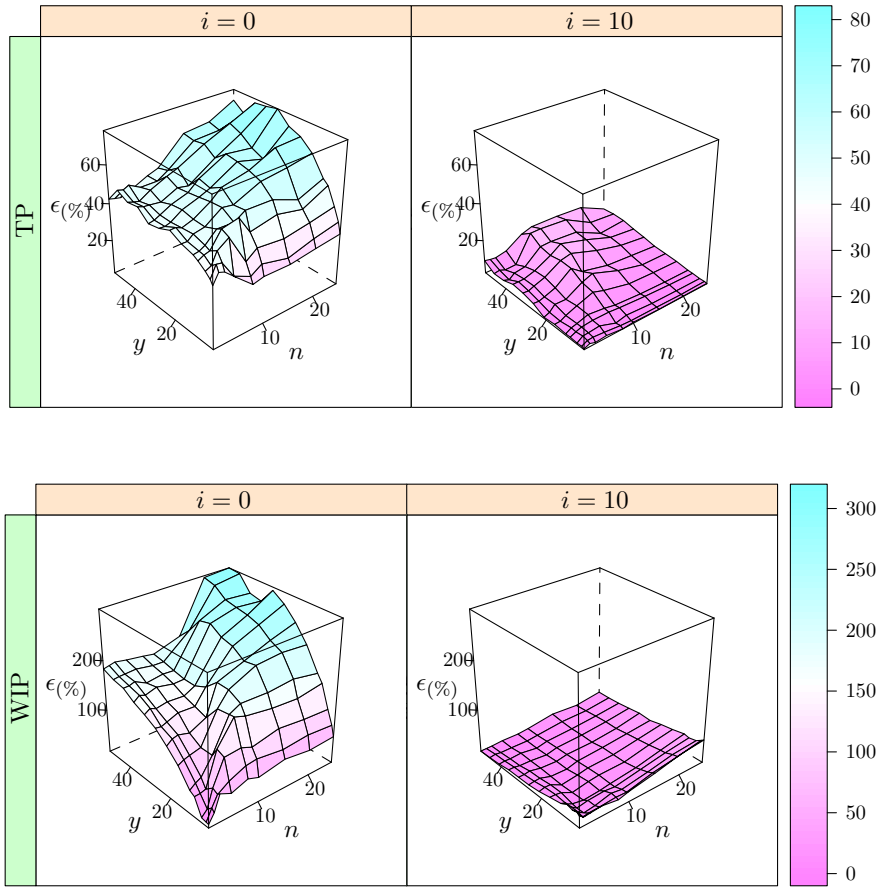


Figure 8.4: The error size in relation to the model size N and resampling rate y and whether a bottleneck was present ($i = 10$) or not ($i = 0$). For ϵ *lower is better* applies.

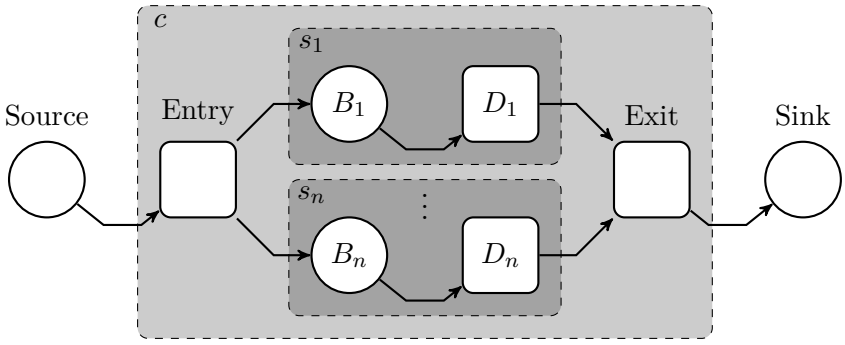


Figure 8.5: The two different coarsening configurations used for validation. The $s_1, \ldots, s_n$ configuration does coarsen all branches separately while the c configuration does coarsen the complete structure.

Things are different when the source is not the bottleneck. In every tested case, other than $i = 0$, the error size was very low compared to the $i = 0$ case. Furthermore, there was no clear relation between the size of the error and the size of n and y. The case $i = 10$ is depicted in Figure 8.4 as an example for all of these cases. In this case ($i = 10$) the source is parameterized with $\mathcal{N}_1$ like most of the machines. Only every tenth machine is parameterized with $\mathcal{N}_2$ which makes it a bottleneck. Due to the high waiting times before the bottlenecks, fluctuations and model dynamics are smoothed out leading to a small error. Of course, in models of size $n < 10$ there are no bottlenecks at all. However, since the source is parameterized the same as the machines (other than in the $i = 0$ case) the probability that a token has to wait in a buffer is relatively high.

This experiment shows, that the two situations, whether a SESE region is starving (a preceding region contains a bottleneck) or the region itself contains a bottleneck and token start to pile up, should be managed differently. Especially in the first case new samples should be taken rather frequently.

8.5 Complete versus Separate

The whole controlling concept is based on the idea that the error increases with the coarsening of larger regions but on the other side saves more runtime. To validate this assumption, a model setup is needed where the material flow can be coarsened in different ways. Therefore, model F was simulated with two different coarsening setups as depicted in Figure 8.5. In the first setup, every branch forms its own region $s_1, \ldots, s_n$ and is separately coarsened. In the second setup, the complete model from *Entry* to *Exit* is coarsened (region c). For both configurations the relative error for all three ratios and the runtime savings were recorded.

Figure 8.6 shows the simulation experiment for the model configuration

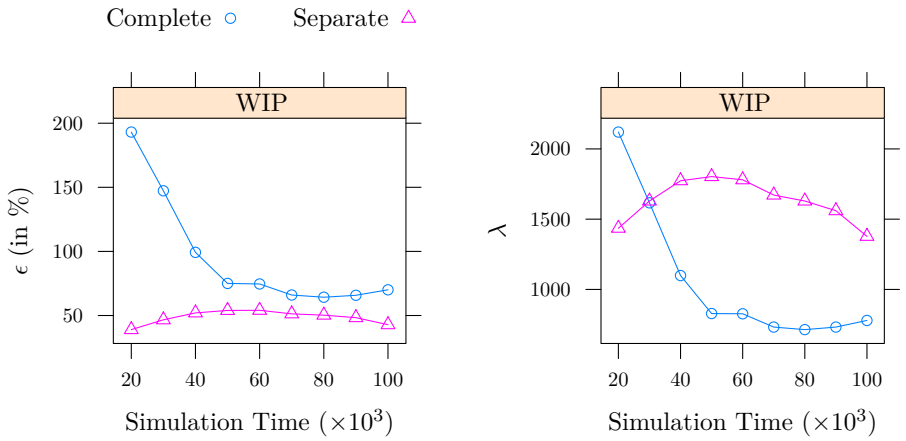


Figure 8.6: This figure shows the output error ϵ and the coarsening efficiency λ for the two different coarsening configuration from Figure 8.5 over time. For both values *The lower, the better* is true.

$F(6, 1, \mathcal{N}(10, 50), \mathcal{N}(100, 50))$. While the previous experiment showed that especially for starving regions (i.e. no bottleneck within the region) the error depends on the resampling rate, this experiment shows the same behavior for the case where $i > 0$. The output difference ϵ is always lower for the separate coarsened setup. This corresponds with the results of the previous experiment where it was shown that the error size decreases with decreasing region sizes. Viewed solely, these results would indicate that the model should be partitioned into a lot of small regions that are coarsened. However, when including the speed-up gained by coarsening the model according to Equation (6.22), the results look a bit different. Then the efficiency λ (which roughly translates to *error per speed-gain*) is better (lower) when coarsening the whole structure (region c). Therefore, the assumption made earlier holds.

8.6 The Effect of the Resampling

One of the most runtime consuming processes within the presented coarsening concept is the so called *resampling*. As described in Chapter 6.1 resampling is used to react to the model dynamics by refreshing the samples used for coarsening. To measure the effect of the resampling one can simply compare the error introduced by the coarsening with resampling *enabled* and *disabled*.

Exemplified, the effect of the resampling is shown for a simulation run for the model setup $Q(3, 0, \mathcal{N}(10, 50), \mathcal{N}(100, 50))$ in Figure 8.7. During simulation a bottleneck is introduced into the material flow at fixed time intervals. The bottleneck is active for a short period of time, after which it

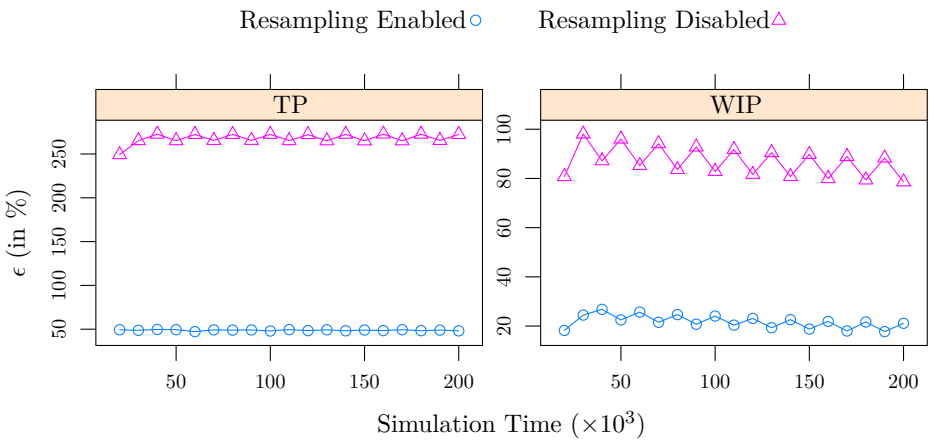


Figure 8.7: The relative error ϵ plotted over time for two different ratios. Two scenarios are shown: The resampling was enabled (blue) or disabled (pink).

is cleared. In such a dynamic environment it is difficult the taken samples become invalid quite fast. Resampling provides a way to gain new samples that better match the current situation in the simulation model. Therefore, when resampling is enabled the relative output error is much smaller. As before, for the *lead time* the error was negligible and the ratio was therefore omitted from Figure 8.7. The zig-zag-pattern found in the chart is a result of the dynamic implementation of the bottleneck into the material flow.

8.7 Subsystem Runtime Consumption

This section explores the runtime consumption of the different subsystems.

8.7.1 Preprocessing and Program Structure Tree Runtime

When applied to a material flow graph in a preprocessing step for each component look-up tables must be setup and the region hierarchy has to be computed. As specified in Chapter 5 the runtime of the preprocessing counts towards the overall runtime of the simulation run. In other words, the preprocessing must be reasonable fast so that over the course of one simulation run the coarsening concept is able to make up for the needed preprocessing runtime. Furthermore, in Chapter 6.4.2 the claim was made that the PST algorithm is fast enough to handle interactive insertion and deletion of material flow components. To validate both claims, the runtime of the PST algorithm to construct the region hierarchy was measured for

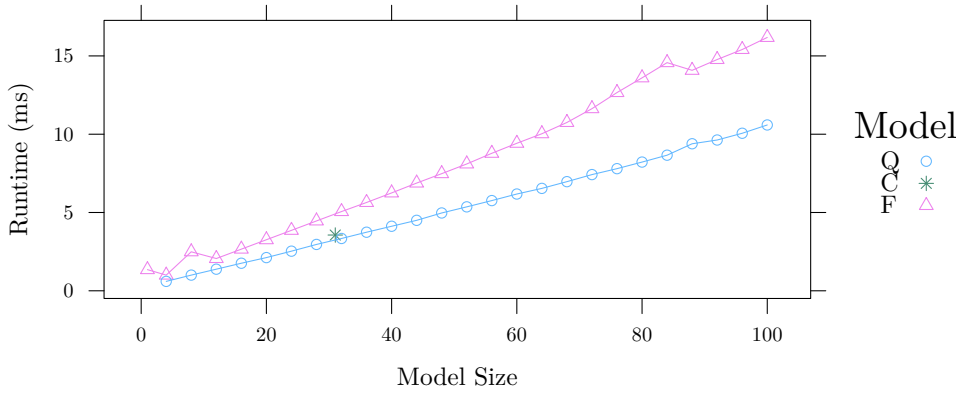


Figure 8.8: The runtime of the PST algorithm (in milliseconds) for three different models. The model A is presented in detail later on. Since the models F, Q are generated, their size was varied. The model size is defined as the number of nodes in the material flow graph.

differently sized simulation models. The runtime to setup of the look-up tables is negligible. The result for three different models is shown in Figure 8.8.

Since the models F, Q are generated on purpose, their size has been varied. One can easily see that the runtime costs linearly increase with the number of material flow components in the model. However, due to the large branching structure, the costs for model F increase faster than for model Q. The model A is taken over from Huber [HD09] and is presented later on in detail (Chapter 8.9.1). The model is much more complex than F, Q and its material flow graph is closer to reality. However, its runtime costs are located between those of the generic models. Thus it can be assumed that the PST runtime costs for other models are about the same order of magnitude. The highest runtime costs of fifteen milliseconds were measured for a model F of size one hundred. In relation to simulation runs that take several minutes to hours, the preprocessing is negligible.

With such a low runtime for a preprocessing step the coarsening concept is even suited for frequent changes of the material flow. For example, it can be used in an interactive material flow editor where the user edits a running material flow simulation. Furthermore, it can adapt to changes in the material flow graph that are triggered by the simulation itself. For example, the material flow graph could change its structure based on optimization algorithms.

Sampling

During runtime samples are taken and stored in look-up tables for proper parameterization of the region replacements. The samples are gathered by observing the simulated material flow system for changes triggered by events from event set (6.14). That means, almost every time an event from (6.14) is triggered some additional computation (6.16) has to be done. To measure the impact on the overall runtime the simulation models $Q(n, 6, \mathcal{N}(10, 5), \mathcal{N}(15, 5))$ with $n := \{10, 20, 30, 50\}$ were simulated with an activated event accumulation but without coarsening. The measured runtime was always around 10% longer than the simulation without the additional event processing.

Bottleneck Identification

For the bottleneck identification and tracking the shifting bottleneck method of Roser et al. [RNT02] has been implemented. The whole identification method is based around periods of time where material flow components such as conveyors and machines are marked as *active*. These intervals have to be managed and overlapping intervals must be found to mark them as *shifting* time periods. Each region maintains its own interval and bottleneck database. Thus, each interval database remains small which keeps the runtime overhead small. The additional computation costs for the bottleneck identification were measured using the same simulation models as for the *Sampling* measurement, discussed in the previous paragraph. The measurements showed that simulations with an active bottleneck detection need around 13% additional runtime.

Conclusion

In conclusion, the token sampling plus the bottleneck detection are responsible for a 23% longer runtime. The coarsening method has to save more than this amount of runtime to be faster than the original simulation model. This raises the question which amount of tokens must be processed by the coarsened regions to clear the addition costs. In other words: When is the break-even point reached? This question is answered in the next section.

8.8 Determining the Break-even Point ω

The break-even is the point in time where the additional runtime costs generated by the coarsening method are compensated by its savings. As explained in the previous section, additional costs are caused by the token sampling to fill the look-up tables, the bottleneck detection and the construction of an appropriate region state when switching between the

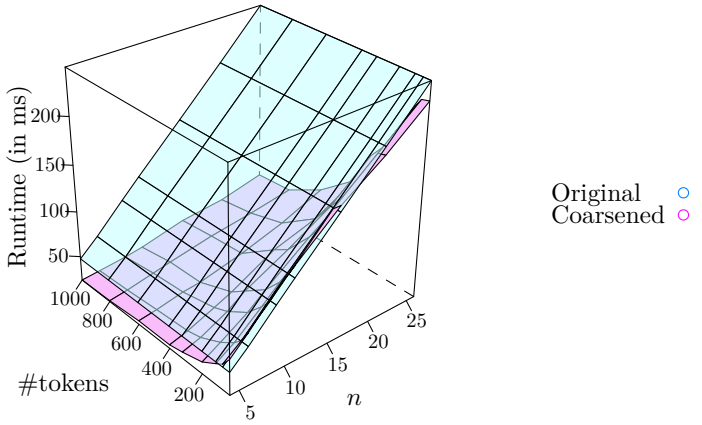


Figure 8.9: In this diagram the absolute runtime for the *original* simulation model and a *coarsened* variant is plotted against varying model sizes (n) and varying number of tokens (#tokens) being coarsened. The break-even point is where both planes intersect.

regions and their replacement. The additional costs and the savings correlate with the number of processed tokens and not with the size or structure of the model. The costs for the look-up table filling and bottleneck detection directly depend on the number of events triggered by tokens (6.14) which have to be processed. Furthermore, the construction of the component states when switching depends on the number of tokens currently located in that region or in the replacement component, respectively. Therefore, a simple model $Q(30, 6, \mathcal{N}(10, 5), \mathcal{N}(15, 5))$ has been used to determine the break-even point.

To compute the break-even point the number of tokens processed with an active coarsening was increased from one to 500 while measuring the overall runtime needed to simulate a fixed time interval (here 200.000 time units). The measured runtime was then compared with the runtime of the original simulation model with the same parameterization. On the test machine the break-even point was measured to be around 40 tokens ($\pm 10\%$ in additional runs). This is shown in Figure 8.9 where the runtime for the original and the coarsened model are plotted against varying model sizes and varying number of tokens being coarsened. The break-even point is where the two depicted planes intersect each other.

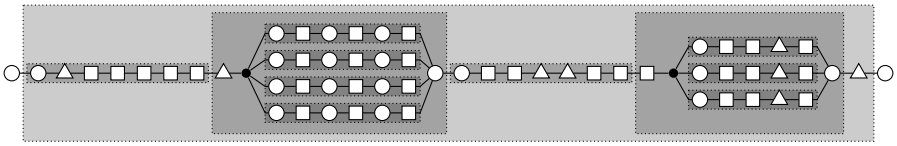


Figure 8.10: Model C. SESE regions are depicted as grey rectangles .

Furthermore, with the determined break-even point it is possible to transform the number of tokens into values of y where $y(\omega) = 1$. Then Figure 8.9 shows the relation between parameter y and the speed-up by model size. It shows, that for small model sizes the relative speed-gain becomes smaller the larger y (or rather #tokens) gets. On the other side, for large model sizes the relative speed-gain remains pretty high for larger y parameters. That means, for the tested simulation model for small model sizes around ten to twenty components large values for y are relatively inefficient. Also, the diagram shows, that around $y = 25 (= 1000$ tokens) the coarsened model only needs ten percent of the runtime of the original simulation model. However, such a low runtime usually also means a very coarsened model computation with a large output difference (the output error) to the output of the original model. In the next section the runtime savings and the output error will be put into relation.

8.9 Evaluation of the Controlling Function

In the previous sections experiments with the purpose-build models Q and F were conducted. These models were used to evaluate the different subsystems of the coarsening method in controlled environments. This was necessary to lower side effects and by that gaining meaningful results. However, to evaluate the controlling function a more complex model with a bigger region hierarchy is needed. Therefore, model C introduced by Huber [Hub09] has been reproduced. It will be covered in short in the next section. After that, the results are presented.

8.9.1 Model C

Model C has a rather simple material flow. It contains machines, buffers of fixed size and conveyors but no assembly or disassembly. The graph and the partitions found by the PST algorithm are depicted in Figure 8.10.

The found regions match the overall logic very well. Especially the parallel structures are put together into one SESE region. This is due to the cleanup step integrated into the modified PST algorithm that combines regions with a single component to larger SESE regions where applicable.

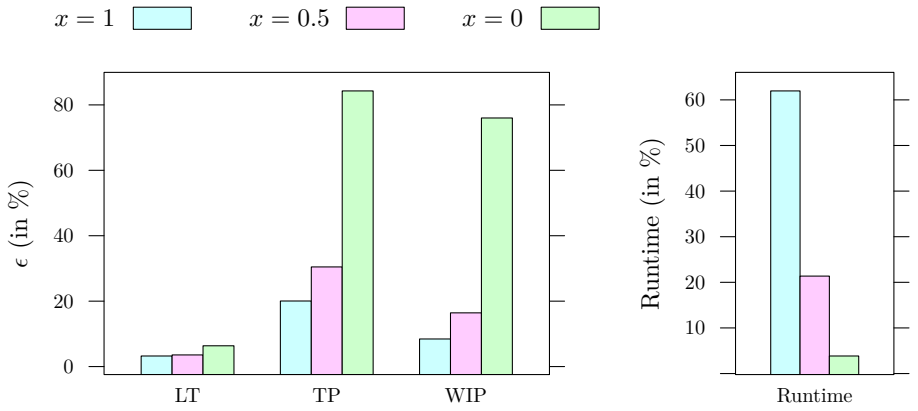


Figure 8.11: This figure shows the (relative) results when coarsening Model C. The results for three different configurations of the controlling mechanism are depicted.

8.9.2 Results

Model C was run with configuration $(i = 20, \mathcal{N}(100, 100), \mathcal{N}(1000, 250))$. This placed one bottleneck in the middle of the second single production line. Furthermore, the variance of the processing times of the machines was higher than in the experiments described so far. The model was simulated for a fixed simulation time of two million time units. At the end, the values of the three ratios and the needed computation time to complete the simulation run were recorded. Four different configurations where simulated: the unaltered, original model and the same model with enabled coarsening in three different configurations, namely $x = 0.5$, $x = 1$ and $x = 0$. Parameter y was set to one. The results are shown in Figure 8.11.

The controlling mechanism shows the intended behavior: For each of the three measured ratios LT and TP and WIP the error increases with a decreasing x while at the same time the runtime gets shorter. As before, the relative error for the *lead time* is the lowest among the measured ratios. However, when setting $x = 0$ the error for the *throughput* and the *work in process* becomes very large (both being around 80% aberration). Also for the two other configurations the relative difference is significant higher than for the *lead time*. This shows that the current parameterization of the region replacements resembles the original *lead time* very well but has problems when it comes to the TP and WIP ratios.

With the $x = 1$ setting the controlling mechanism only activates the coarsening for serial connected regions and on low variances. Therefore, this setting shows the lowest relative error for all the ratios but still needs more

than 60% of the runtime of the original simulation model. Setting $x = 0.5$ shows the intended trade off: It has a significant lower runtime but also a higher relative error especially for TP and WIP. The $x = 0$ setting tries to maximize the efficiency (low error per speed up) (see Equation 6.22). It mostly uses regions higher in the region hierarchy which unfortunately results in very high differences for the TP and WIP ratio. Thus, for TP and WIP this setting is not very efficient. Instead, for the three parameterizations of x, $x = 0.5$ is the most efficient one. In this case, the controlling mechanism doesn't work as expected.

8.10 Conclusion

If applied to a given material flow simulation model, the proposed coarsening method is able to save runtime and to control the relative output error, based on user preferences. purpose-build models were used to shown that the different subsystems of the coarsening method work reasonable well. These models were also used to understand which parameters affect the runtime and overall error.

The coarsening method works great when solely measuring the *lead time* of the tokens. In all experiments the relative output error was always below four percent. In this case, the coarsening could be setup to aggressively reduce the runtime with a large y parameter value and $x = 0$ since the output error will remain very low. However, the method does not work that good when it comes to other ratios like the token *throughput* or the *work in process*. Especially for the latter relative errors up to 250% have been measured in certain configurations.

However, in the end the output error depends especially on the model dynamics. For example, if a simulated production line shows chaotic behavior with largely different processing times, then samples can be taken as frequently as possible, they still do not resemble the processing times of the original model. On the other side, models that have a poorly constructed material flow with a lot of redundant computations the coarsening method would work quite well. Therefore, the user should specify the x, y-parameters in dependency of the model that will be coarsened. That means, the user still needs some knowledge about the simulation model (especially about its dynamics) to use the coarsening method in an optimal and efficient way.

Chapter 9

Conclusion

In this thesis a coarsening method has been presented that is designed to be applicable to (almost) arbitrary material flow models. The *Token Sampling* concept overcomes several drawbacks present in previous coarsening methods:

- The material flow system specifications developed in this thesis leave the high-level processes intentionally undefined. Therefore, the coarsening method can be used with arbitrary token processing systems.
- Most notably the coarsening method does not need a computational and time intensive preprocessing. Instead its preprocessing takes only milliseconds for reasonable sized models (see Chapter 8.7.1). Thus it can be applied during runtime at will and it is fast enough to adapt itself to dynamic changes of the material flow structure. This allows the usage of the coarsening method even during model construction.
- Since this coarsening method is able to handle arbitrarily connected SESE regions it does not depend on the presence of certain structures. Instead, almost any simulation model structure can be coarsened in some way.
- Due to the utilization of the *shifting bottleneck* detection method the coarsening method is able to adapt itself to changing model states. Especially sudden break downs of machines are tracked and considered when choosing which part of a simulation model to coarsen.

The controlling of the coarsening method was designed to be understandable and user-friendly. Model and simulation system dependent parameters have been avoided and the user does not have to specify parameters that are only understandable with deep knowledge of the coarsening and simulation

method. The user simply provides two parameters x and y. The former specifies a trade-off between efficiency and error avoidance. The latter defines a maximum amount of tokens when coarsening regions - in terms of break-evens. All other (internal) parameter are determined by the coarsening method itself.

The practical results show that the overall concept works very well for the *lead time* ratio (around $\pm 4\%$ difference from the simulation results of the original model). However, the other two ratios *throughput* and *work in process* usually have a much higher difference. The region replacements used for the coarsening imitate the *lead time* behavior very well but need some work to lower the difference for the other two ratios.

Under-Researched

While the presented concept works reasonable well it also has some room for improvement. Especially the behavior of the region replacement components could be updated to reduce the output error for ratios like the *throughput* and *work in process*. For example, instead of simply setting the size of the replacement to the size of the replaced region (6.18) the size of the replacement could be set to a mean value of the work in process of that region. This could improve (lower) the output difference at least for the *work in process* ratio.

Furthermore, the concept could be adapted to generic input-output-system descriptions like DEVS. It would be possible to implement the look-up tables in such a way that (almost) generic input-output-relations could be sampled. Using the PST algorithm SESE regions could be identified in the graph of connected DEVS systems which then could be replaced. This would allow the coarsening of generic DEVS systems and would improve an earlier work of Sevinc [Sev90] which also needs a very computational intensive preprocessing. Adapting this concept would omit the preprocessing and make the method of generic model abstraction much more practical.

Bibliography

[Bar98] Russell R. Barton. Simulation metamodels. In *Proceedings of the 1998 Winter Simulation Conference*, volume 1, pages 167–174, 1998.

[BH97] Kim Breitfelder and Stephen Huffman, editors. *The IEEE Standard Dictionary of Electrical and Electronics Terms*. Institute of Electrical and Electronics Engineers, Inc, sixth edition edition, 1997.

[BT96] Roger J. Brooks and Andrew M. Tobias. Choosing the best model: Level of detail, complexity, and model performance. In *Mathematical and Computer Modelling*, volume 24, pages 1–14. Elsevier Science Ltd, 1996.

[BT00] Roger J. Brooks and Andrew M. Tobias. Simplification in the simulation of manufacturing systems. *International Journal of Production Research*, 38:1009–1027(19), March 2000.

[CBP00] Leonardo Chwif, Marcos Ribeiro Pereira Barretto, and Ray J. Paul. On simulation model complexity. In *Proceedings of the 2000 Winter Simulation Conference*, pages 449 – 455, 2000.

[CL06] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Com90] CompuServe Incorporated. Graphics interchange format(sm) version 89a. World Wide Web Consortium, July 1990.

[CPB06] Leonardo Chwif, Ray J. Paul, and Marcos Ribeiro Pereira Barretto. Discrete event simulation model reduction: A causal approach. *Simulation Modelling Practice and Theory*, 14(7):930 – 944, 2006.

[Dea05] John Deacon. *Object-Oriented Analysis and Design.* ADDISON-WESLEY, 2005.

[Deu94] Deutsche Elektrotechnische Kommission im DIN und VDE (DKE). Norm 19226-1: Regelungstechnik und steuerungstechnik - allgemeine grundbegriffe. Technical report, Deutsches Institut für Normung e.V., Berlin, Februar 1994.

[DH93] Paul K. Davis and Richard Hillestad. Families of models that cross levels of resolution: Issues for design, callibration and management. In *Proceedings of the 1993 Winter Simulation Conference*, pages 1003 – 1012, 1993.

[Ebe01] David H. Eberly. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics.* Morgan Kaufmann, 2001.

[EJF10] Benjamin Eikel, Claudius Jähn, and Matthias Fischer. Preprocessed global visibility for real-time rendering on low-end hardware. In *Advances in Visual Computing*, volume 6453 of *Lecture Notes in Computer Science*, pages 622–633. Springer, Berlin / Heidelberg, 2010.

[EJP11] Benjamin Eikel, Claudius Jaehn, and Ralf Petring. Padrend: Platform for algorithm development and rendering. In Jürgen Gausemeier, Michael Grafe, and Friedhelm Meyer auf der Heide, editors, *Augmented & Virtual Reality in der Produktentstehung*, volume 295 of *HNI-Verlagsschriftenreihe, Paderborn*, pages 159–170. Heinz Nixdorf Institut, Universität Paderborn, May 2011.

[Fel13] Andreas Emil Feldmann. Fast balanced partitioning is hard even on grids and trees. *Theoretical Computer Science*, 2013.

[FF85] Linda W. Friedman and Hershey H. Friedman. Validating the simulation metamodel: Some practical approaches. *SIMULATION*, 45(3):144–146, September 1985.

[Fis89] Paul A. Fishwick. Neural network models in simulation: a comparison with traditional modeling approaches. In *WSC '89: Proceedings of the 21st conference on Winter simulation*, pages 702–709, New York, NY, USA, 1989. ACM.

[Flo95] John E. Flood. *Telecommunications Switching, Traffic and Networks.* Prentice Hall, 1995.

[Fra95] Frederick K. Frantz. A taxonomy of model abstraction techniques. In *WSC '95: Proceedings of the 27th conference on Winter simulation*, pages 1413–1420, Washington, DC, USA, 1995. IEEE Computer Society.

[Fri03] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Computer Society Pr, 2003.

[FRL+10] Matthias Fischer, Hendrik Renken, Christoph Laroque, Guido Schaumann, and Wilhelm Dangelmaier. Automated 3d-motion planning for ramps and stairs in intra-logistics material flow simulations. In *Proceedings of the 2010 Winter Simulation Conference (WSC 2010)*, pages 1648 – 1660. IEEE, Omnipress, December 2010.

[Fuj98] Richard M. Fujimoto. Parallel and distributed simulation. In Jerry Banks, editor, *Handbook of Simulation*, pages 429 – 464. John Wiley & Sons, 1998.

[Gre09] Jason Gregory. *Game engine architecture.* A K Peters, first edition, April 2009.

[HD09] Daniel Huber and Wilhelm Dangelmaier. Controlled simplification of material flow simulation models. In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, editors, *Proceedings of the 2009 Winter Simulation Conference*, 2009.

[Hel04] Helsinki University of Technology. *Lecture Notes: S-38.145 - Introduction to Teletraffic Theory*, 2004.

[HL99] Yi-Feng Hung and Robert C. Leachman. Reduced simulation models of wafer fabrication facilities. *International Journal of Production Research*, 37:2685–2701, August 1999.

[HRS+07] Christian Henke, Carsten Rustemeier, Tobias Schneider, Joachim Böcker, and Ansgar Trächtler. Railcab - Ein Schienenverkehrssystem mit autonomen, Linearmotor-getriebenen Einzelfahrzeugen. In *ETG-Fachbericht-Internationaler ETG-Kongress 2007.* VDE VERLAG GmbH, 2007.

[Hub09] Daniel Huber. *Geregelte Vereinfachung hierarchischer Partitionen von Modellen in der Materialflusssimulation.* PhD thesis, Universität Paderborn, 2009.

[Inc09] Incontrol Simulation Software. Tutorial ed 8, 2009.

[Inc12a] Incontrol Simulation Software. Enterprise dynamics. Accessed April, 2012. `http://www.incontrolsim.com`, April 2012.

[Inc12b] Incontrol Simulation Software. Enterprise dynamics technical overview. Accessed April, 2012. `http://www.incontrolsim.com/en/ed-platform/technical-overview.html`, April 2012.

[JFM05] Rachel .T. Johnson, John.W. Fowler, and Gerald.T. Mackulak. A discrete event simulation model simplification technique. In *Simulation Conference, 2005 Proceedings of the Winter*, page 5 pp., 2005.

[JLGL99] Sanjay Jain, Chu-Cheow Lim, Boon-Ping Gan, and Yoke-Hean Low. Criticality of detailed modeling in semiconductor supply chain simulation. In *WSC '99: Proceedings of the 31st conference on Winter simulation*, pages 888–896, New York, NY, USA, 1999. ACM.

[JPP94] Richard Johnson, David Pearson, and Keshav Pingali. The program structure tree: computing control regions in linear time. In *Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*, PLDI '94, pages 171–185, New York, NY, USA, 1994. ACM.

[KLM96] C-T Kuo, J-T Lim, and SM Meerkov. Bottlenecks in serial production lines: A system-theoretic approach. *Mathematical Problems in Engineering*, 2(3):233–276, 1996.

[LB95] Stephen R. Lawrence and Arnold H. Buss. Economic analysis of production bottlenecks. *Mathematical Problems in Engineering*, 1(4):341–363, 1995.

[LCN09] Lin Li, Qing Chang, and Jun Ni. Data driven bottleneck detection of manufacturing systems. *International Journal of Production Research*, 47(18):5019–5036, 2009.

[LK00] Averill M. Law and David W. Kelton. *Simulation Modelling and Analysis*. McGraw-Hill Education, April 2000.

[Mer05] Galina Merkuryeva. Metamodelling for simulation applications in production and logistics. Technical report, Department of Modelling and Simulation, Riga Technical University, 2005.

[MT75] M. D. Mesarovic and Y. Takahara. *General Systems Theory: Mathematical Foundations*, volume 113 of *Mathematics In Science And Engineering*. Academic Press, 1975.

[Mue05] Bengt Mueck. *Eine Methode zur benutzerstimulierten detaillierungsvarianten Berechnung von diskreten Simulationen von Materialflüssen*. PhD thesis, Universität Paderborn, 2005.

[MVA10] Jeff McAffer, Paul VanderLei, and Simon Archer. *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley, Upper Saddle River, NJ, 2010.

[Nie77] Gerhard Niemeyer. *Kybernetische System- und Modelltheorie: system dynamics*. Verlag Franz Vahlen, 1. aufl. edition, 1977.

[Pic75] Franz Pichler. *Mathematische Systemtheorie: Dynamische Konstruktionen*. de Gruyter, 1975.

[PN03] Sachin B Patkar and H Narayanan. An efficient practical heuristic for good ratio-cut partitioning. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 64–69. IEEE, 2003.

[PR08] Carl Adam Petri and Wolfgang Reisig. Petri net. http://www.scholarpedia.org/article/Petri_net, 2008.

[RD13] Hendrik Renken and Wilhelm Dangelmaier. Improving flow-based modeling of enterprise systems and modeling of custom warehouse systems in d^{3}fact. In *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 94–101. INSTICC, INSTICC PRESS, Jul 2013.

[REK12] Hendrik Renken, Felix Alexander Eichert, and Alexander Klaas. Visualization and collaborative editing of simulation models with heterogeneous clients - implemented into the simulator d^{3}fact. In *32nd Computers and Information in Engineering Conference*, volume 2. ASME, 2012.

[Ren11] Hendrik Renken. d^{3}fact projekt wiki. Accessed Feb. 1, 2011. `https://macabeo.cs.upb.de/trac/d3fact/`, March 2011.

[RNT02] Christoph Roser, Masaru Nakano, and Minoru Tanaka. Shifting bottleneck detection. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 2, pages 1079–1086. IEEE, 2002.

[RNT03] Christoph Roser, Masaru Nakano, and Minoru Tanaka. Comparison of bottleneck detection methods for agv systems. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 2, pages 1192–1198. IEEE, 2003.

[Roc11] Rockwell Automation, Inc. Arena simulation software by rockwell automation. Accessed Feb. 1, 2011. `http://www.arenasimulation.com/`, January 2011.

[Rop79] Günter Ropohl. *Eine Systemtheorie der Technik - Zur Grundlegung der Allgemeinen Technologie*. Carl Hanser Verlag München Wien, 1979.

[Rop09] Günter Ropohl. *Allgemeine Technologie : eine Systemtheorie der Technik*. Universitätsverlag Karlsruhe, 3. aufl. edition, 2009.

[Ros99] Oliver Rose. Estimation of the cycle time distribution of a wafer fab by a simple simulation model. In *Proceedings of the SMOMS 1999*, pages 133–138, 1999.

[Ros00] Oliver Rose. Why do simple wafer fab models fail in certain scenarios? *Winter Simulation Conference*, 2:1481–1490, 2000.

[Ros07] Oliver Rose. Improved simple simulation models for semiconductor wafer factories. In *Proceedings of the 2007 Winter Simulation Conference*, pages 1708–1712, 2007.

[SDV08] Sankar Sengupta, Kanchan Das, and Robert P VanTil. A new method for bottleneck detection. In *Proceedings of the 40th Conference on Winter Simulation*, pages 1741–1745. Winter Simulation Conference, 2008.

[Sev90] Suleyman Sevinc. Automation of simplification in discrete event modelling and simulation. *International Journal of General Systems*, 18(2):125–142, 1990.

[Sev91] Suleyman Sevinc. Theories of discrete event model abstraction. In *WSC '91: Proceedings of the 23rd conference on Winter simulation*, pages 1115–1119, Washington, DC, USA, 1991. IEEE Computer Society.

[SF98] Alex F. Sisti and Steven D. Farr. Model abstraction techiques: An intuitive overview. In *Aerospace and Electronics Conference, 1998. NAECON 1998. Proceedings of the IEEE 1998 National*, pages 447 – 450. IEEE Computer Society, 1998.

[SG96] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[Sof11] Wolverine Software. Wolverine web. Accessed Feb. 1, 2011. `http://www.wolverinesoftware.com/`, January 2011.

[Som04] Ian Sommerville. *Software Engineering*. Addison Wesley, seventh edition, 2004.

[SSL+01] Rajarishi Sinha, Rajarishi Sinha, Vei-Chung Liang, Student Member, Christiaan J. J. Paredis, and Pradeep K." Khosla. Modeling and simulation methods for design of engineering systems. *JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING*, 1:84–91, 2001.

[SY93] Lee Schruben and Enver Yücesan. Complexity of simulation models: a graph theoretic approach. In *WSC '93: Proceedings of the 25th conference on Winter simulation*, pages 641–649, New York, NY, USA, 1993. ACM.

[VB05] Wim C. M. Van Beers. Kriging metamodeling in discrete-event simulation: an overview. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, pages 202–208. Winter Simulation Conference, 2005.

[VDI96] Richtlinie 3633: Simulation von logistik-, materialfluß- und produktionssystemen - begriffsdefinitionen, November 1996.

[VDI08] Richtlinie 2689: Leitfaden für materialflussuntersuchungen, April 2008.

[VG03] Sven Völker and Peter Gmilkowsky. Reduced discrete-event simulation models for medium-term production scheduling. *Systems Analysis Modelling Simulation*, 43(7):867–883, 2003.

[Wal87] Jack C. Wallace. The control and transformation metric: Torward the measurment of simulation model complexity. *Proceedings of the Winter Simulation Conference*, page 597 ff., 1987.

[Wym93] A. Wayne Wymore. *Model-based Systems Engineering.* CRC Press, 1993.

[WZZ05] Yongcai Wang, Qianchuan Zhao, and Dazhong Zheng. Bottlenecks in production networks: An overview. *Journal of Systems Science and Systems Engineering*, 14(3):347–363, 2005.

[XJ 12a] XJ Technologies Company. Discrete event simulation modeling tool. Accessed April, 2012. `http://www.xjtek.com/anylogic/approaches/discreteevent/`, May 2012.

[XJ 12b] XJ Technologies Company. Why anylogic simulation software? Accessed April, 2012. `http://www.xjtek.com/anylogic/why_anylogic/`, April 2012.

[Zei76] B. P. Zeigler. *Theory of Modeling and Simulation.* Wiley Interscience, 1976.

[ZP00] Bernard P. Zeigler and Herbert Praehofer. *Theory of Modeling and Simulation.* Academic Press, January 2000.

Appendix A

Glossary

I A possibly infinite index set. 6, 66

P_S The set of spaces of a system $S \in \mathcal{S}_\mathcal{K}$. 66

P A set of places $P := \{p_i \mid i \in I\}$ that can store tokens $k \in \mathcal{K}$. 137

Φ A (recursive) function that specifies which regions are coarsened, based on the λ measurement. 91, 98

Ψ A function that specifies how long a region will be coarsened, based on the expected output error ϵ and the user specified reference output x. 93, 95, 99

$\check{E}_{i_S}$ A sequence of observed events for a specific token processing system S. 73, 74

$\check{E}_{i_k}$ A sequence of observed events for a specific token k. 73–77, 82, 83

$\check{E}_i$ A sequence of observed events $\check{E}_i := \{\ \check{e}_0, \check{e}_1, \ldots\}$. 77

$\check{\bar{a}}_p$ Event indicating the *blocking* of a space $p \in P$. 72, 74

$\check{\bar{a}}$ Event indicating the *blocking* of a space. 75–77

$\check{\bar{q}}_p$ Event indicating that a space $p \in P$ now no longer contains a token. 72–75

$\check{\bar{q}}$ Event indicating that a space now no longer contains a token. 76, 77

$\check{a}_p$ Event indicating the *accessibility* of a space $p \in P$. 72, 74

$\check{a}$ Event indicating the *accessibility* of a space.. 75–77

$\check{q}_p$ Event indicating that a space $p \in P$ now contains a token $k \in \mathcal{K}$. 72–75, 77

$\check{q}$ Event indicating that a space now contains a token. 75–77

ϵ This specifies the difference between the simulation output $Y(M), Y(M')$ of two models M, M'. If one of the models is coarsened ϵ is termed *output error*. 90–96, 98, 99, 114, 116–119

$\hat{\epsilon}$ This specifies the *expected* difference between the simulation output $Y(M), Y(M')$ for two models M, M'. 99

$\hat{\mu}$ This specifies the *expected* speed gain for a coarsened region r. 99

λ The coarsening efficiency describes the expected output error by the speed gain for a coarsened region r. 90–92, 106, 117, 118

$[W]$ is the *look-up table* for recorded paths taken by tokens through a region in a material flow system. 87, 88, 99, 108, 144

$[lt]$ is the *look-up table* for the lead time. 82, 83, 99, 106

$[\bullet]$ is the *look-up table* for the waiting time. 82–84, 105, 143

$[\otimes]$ is the *look-up table* for the processing time. 77, 82–84, 105, 143

$\mathcal{C}_\mathcal{K}$ A set of tuples $E_K \subseteq \mathcal{S}_\mathcal{K} \times \mathcal{S}_\mathcal{K}$ indicating a connection between two systems. 65, 68, 138

$\mathcal{K}$ The set of all possible tokens. 7, 8, 65–69, 72–74, 84, 85, 101, 137, 138

$\mathcal{R}$ The hierarchy (tree) of identified *single entry single exit* regions of a material flow system. 90, 92, 93

$\mathcal{S}_{XY}$ The class of connectable systems. 23, 65

$\mathcal{S}_\mathcal{K}$ The class of connectable systems that process tokens ($\mathcal{K}$) in some way. 65–67, 69, 137, 138

$\mathcal{T}$ Identifies a triple $(T, \leq, t_0)$ that is interpreted as an ordered set of points in time. 10, 57, 76, 83

$\mathcal{U}$ Identifier for the *state space* of a system. 67, 69, 72

$\mathring{\mathcal{C}}$ The subset of *enabled* channels of $\mathcal{C}_\mathcal{K}$. 69, 70

$\mathtt{r}$ A single component that represents a *coarsened* version of a region r.. VI, 81–88, 93, 95, 106–108, 142, 144

μ This specifies the real world speed gain for a coarsened region r. 90–93, 98

ω A value that specifies the break-even in terms of runtime costs for a coarsened region. 95, 121

ψ A function that maps an arbitrarily large output error ϵ to the interval $[0..1[$. 94, 95

σ The statistical variance of a set of values. 99

$\varnothing$ The arithmetical mean of a set of values. 99

b Identifies the *processing* state of a token. 75, 76

f Identifies the *finished* state of a token. 75, 76

l Identifies the *idle* state in a token. 75, 76

r A region of the material flow graph with connected token processing systems. VI, 81–88, 90–93, 95, 98, 99, 106–108, 142–144

v Identifies the *preprocessing* state of a token. 75, 76

w Identifies the *waiting* state of a token. 75, 76

Appendix B

Listings

This appendix holds all program listings used throughout the thesis. The listings are ordered according to their first reference.

```
CYLCE_EQUIV(G)
    #perform an undirected depth-first search
    for each node n in reverse depth-first order
    {
        # compute hi_n
        min{ dfsnum_t | (n,t) is a back edge } -> hi0
        min{ hi_c | c is a child of n } -> hi1
        min{ hi0, hi1 } -> hi_n
        any child c of n where hi_c = hi1 -> hichild
        min{ hi_c | c is a child of n other than hichild } -> hi2

        # compute bracketlist
        for each child c of n
            bracketlist_n + bracketlist_c -> bracketlist_n

        for each capping back edge d from a descendant of n to n
            remove(b) -> bracketlist_n

        for each back edge b from a descendant of n to n
        {
            remove(b) -> bracketlist_n
            if class_b is undefined
                start new class -> class_b
        }
        for each back edge e from n to an ancestor of n
            push(e) -> bracketlist_n

        if ( hi2 < hi1 )
        {
            # create capping back edge
            edge(n, node(hi2)) -> d
            push(d) -> bracketlist_n
        }

        # determine class for edge from parent_n to n
        if (n is not the root of dfs tree)
        {
            #let e be the tree edge from parent_n to n;
            top(bracketlist_n) -> b
            if ( size_b ≠ |bracketlist_n| )
            {
                |bracketlist_n| -> size_b
                start new class -> tmp-class_b
            }
            tmp-class_b -> class_e

            #check for e,b equivalence
            if ( size_b = 1 )
                class_e -> class_b
        }
    }
```

Listing B.1: The cycle equivalence algorithm used to construct the *program structure tree* in Johnson et al. [JPP94].

```
#p is the source process and u the update
UPDATE(p, u)
    put(u) -> stack

    #inform processes in front of p about u
    0 -> i
    peek(stack) -> u
    while (P[i] ≠ p)
    {
        u -> inform(P[i])
        i+1 -> i
    }

    #do not inform p about u
    if (|stack| > 1)
    {
        #merge u with previous update
        poll(stack) -> u
        poll(stack) -> v
        u + v -> u
        put(u) -> stack
    }
    else
    {
        #there is only one update left on the stack
        i+1 -> i
        peek(stack) -> u
        while (i < |P|)
        {
            u -> inform(P[i])
            i+1 -> i
        }
        poll(stack) #clear stack
    }
```

Listing B.2: The update algorithm for a set of material flow processes at a location.

```
SWITCH_TO_REPLACEMENT(r)
    for (i in |r|-1 to 0)
    {
        #retrieve tokens from i-th TPS
        take(P_{S_i}) -> K
        for (k in K)
        {
            #put every token into replacement
            #but with unique delay time
            put(k, Δt_k) -> P_r
        }
    }
```

Listing B.3: The algorithm for the state transfer from the original components to the replacement. It is used for both sequentially and arbitrarily connected regions.

```
SWITCH_TO_ORIGINAL(r)
    #retrieve tokens from replacement
    take(P_r) -> K

    TRANSFER_TO_ORIGINAL(r,K)
```

Listing B.4: The algorithm for the state transfer from the replacement back to the original components in a sequentially connected region. It utilizes the `TRANSFER_TO_ORIGINAL` subroutine from Listing B.5.

```
TRANSFER_TO_ORIGINAL(S_0, S_1, ..., K)
    |K|-1 -> j
    0 -> mt
    for each (S in reverse order)
    {
        #do not take last waiting time into account
        if (i ≠ |r|-1)
            mt + [●]_S -> mt

        #do take factor x into account
        mt + [⊗]_S * (1-x) -> ct
        j -> h
        #put token into i-th system
        #into post-processing state
        PUT(K, j, h, ct, ●) -> S

        [⊗]_S + mt -> mt
        #put token into i-th system
        #into pre-processing state
        PUT(K, j, h, mt, ⊗) -> S
    }

    #if there are tokens left over
    #search for empty places
    for each (S in reverse order) and while (j >= 0)
    {
        while (P_S not full) and while (j >= 0)
        {
            put(K[j]) -> P_S
            j-1 -> j
        }
    }


PUT_S(K, j, h, t, ○)
#put tokens from K into the storage of system S
#set the tokens to state ○
    while (j >= 0 & j-h < |P_S| & Δt_K[j] < t)
    {
        put(K[j]) -> P_S as ○
        j-1 -> j
    }
```

Listing B.5: The algorithm for the transfer of a set of tokens `K` back to the original components $S_0, S_1, \ldots$, which are specified as a sequence.

```
SWITCH_TO_ORIGINAL(r)
    #retrieve tokens from replacement
    take(P_r) -> K

    for each W in {[W]_r}
    {
        count of W in [W]_r / |[W]_r| -> frac
        min{|W|, frac, |K|} -> size

        random subset of size of K -> k
        K - k -> K

        TRANSFER_TO_ORIGINAL(W, k)
    }

    all paths not full -> 𝒲
    #if there are tokens left over
    #put into random path
    while (|K| > 0)
    {
        random path from 𝒲 -> W

        count of W in [W]_r / |[W]_r| -> frac
        min{|W|, frac, |K|} -> size

        random subset of size of K -> k
        K - k -> K

        TRANSFER_TO_ORIGINAL(W, k)

        if (W is full)
            𝒲 - W -> 𝒲
    }
```

Listing B.6: The algorithm for the state transfer from the replacement back to the original components in a n arbitrarily connected region. It utilizes the `TRANSFER_TO_ORIGINAL` subroutine from Listing B.5.

Appendix C

Large PST Example

This appendix contains a (relatively) large example of a material flow graph on which the modified PST algorithm from Listing B.1 was applied. Due to space constraints on this page, first the result of the PST algorithm is presented (Figure C.1) and afterwards on the next page the example material flow graph (Figure C.2).

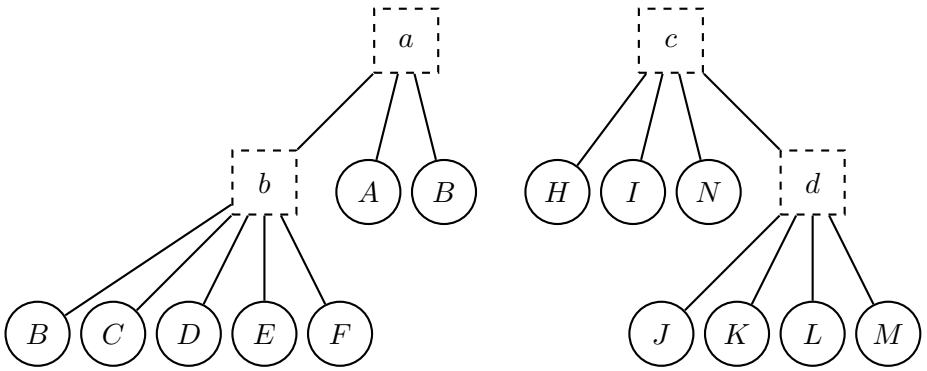


Figure C.1: This is the resulting hierarchy tree. The dashed nodes represent regions and the circle nodes represent the components in the graph.

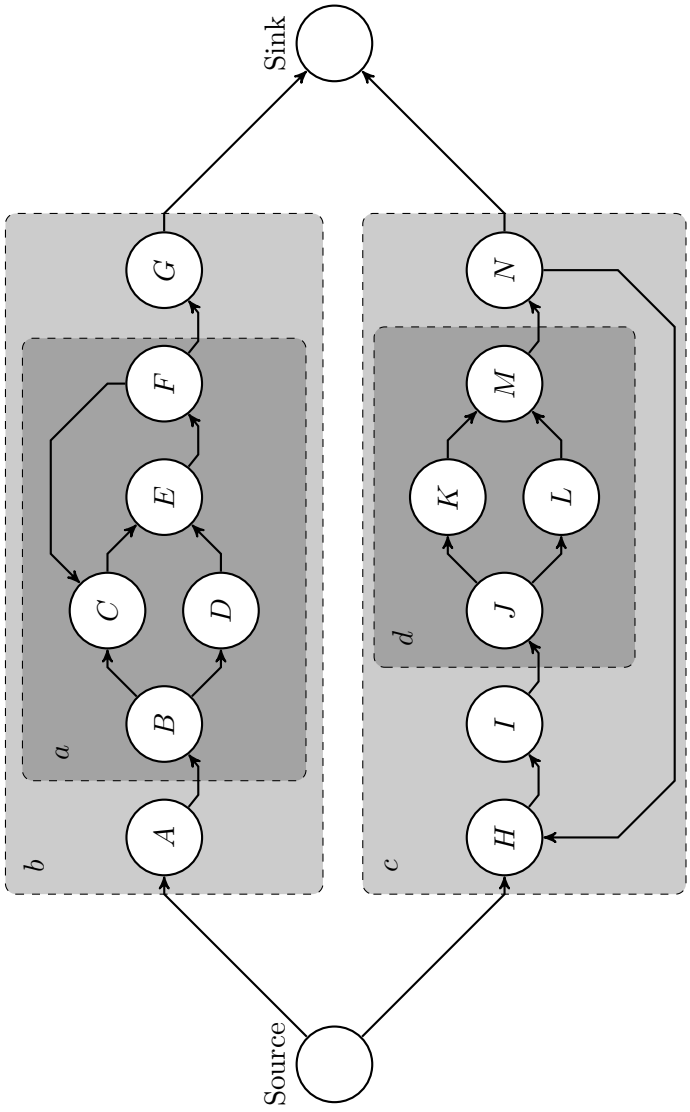


Figure C.2: A (relatively) large material flow graph with back edges. The PST algorithm has found four SESE regions (dashed rectangles).