Simulation Based Parameter and Structure Optimisation of Discrete Event Systems

Olaf Hagendorf

A thesis submitted in partial fulfilment of the

requirements of Liverpool John Moores University

for the degree of Doctor of Philosophy

JMU

Liverpool John Moores University

May 2009

The following figures and appendix has been omitted on request of the university –

Acknowledgements

It seems impossible to reach the end of this long process without the support from many others, who have helped me so much along the way.

Contents

List of Figures

List of Coding Examples

List of Tables

Chapter 1

Introduction

1.1 Preamble

Often it is of interest to study a system to understand the relations between its components or to predict how a system is responsive to changes. Sometimes it is possible to directly experiment with the system. However, this is not always possible e.g. due to costs when a manufacturing system has to be stopped, changed or extended. Often the system even does not yet exit. A model, defined as a representation of the system in order to investigate it, can solve this dilemma. Generally, it is sufficiently to abstract the system with a view to the analysing the issues under investigation. In terms of modelling and simulation this abstract is named the simulation model.

A system can be classified into discrete or continuous: "Few systems in practice are wholly discrete or continuous; but since one type of change predominates for most systems, it will usually be possible to classify a system as being either discrete or continuous." [25]. The analysing issue also plays a decisive role. An analogue printer in a photofinishing lab is a typical example. It is possible to analyse the machine at a very low level with the continuous movements of machine components and analogue film material when the objective is to optimise the component interaction. Another, discrete viewpoint could be the number of pictures and the length of photographic paper handled in a specific amount of time when the objective is to plan throughput and the necessary staff.

Simulation models as a particular type of mathematical system models can be classified too, e.g. as being static or dynamic, deterministic or stochastic, and discrete or continuous. A static simulation model represents a system at a particular time whereas a dynamic simulation model represents system changes over time. A deterministic simulation model does not contain any random variables whereas a stochastic simulation model has in minimum one random variable as an input. Discrete and continuous models can be discrete and continuous systems as described above. One specific type of discrete systems is the discrete event system (DES) where state variables change at discrete points in time during simulation.

One of the most important applications of modelling and simulation based on discrete event systems are manufacturing systems. These systems have been modelled since the origins of manufacturing. From the civilisations of the ancient world to the first industries through to current high-technology production, managers and engineers have thought about the complexities of manufacturing systems [27]. As computers developed they became an increasing important means of modelling and simulation. The expanding capability of computing systems and the increasing demands of engineers and managers planning, implementing and maintaining manufacturing systems have been pushing the boundaries of modelling and simulation research. With the decreasing costs of computing systems, modelling and simulation applications have become an integral part of industrial practice.

Simulation has been used widely and successfully to support the design of new production facilities and material handling systems and to evaluate variants of existing systems. Applications for production, warehouse-management and material handling control can incorporate simulation techniques to evaluate staffing and operating rules, changes of material handling and system layout or the effect of capital investment. An important advantage in using modelling and simulation techniques is the possibility of evaluating changes before making investment decisions and without disturbing the existing system.

Recently, with increasing globalisation, the competition conditions for manufacturing have been changing fundamentally. A key shift is the need to move from increasing product quantity to a combination of increasing quantity and a drive for manufacturing flexibility. As the number and the speed of product innovations increase, the time to market and the marketing life of a product decreases. As a consequence manufacturers have to extend the general objective "cost saving" to "time and cost saving" [29]. To support this market trend manufacturing systems will increase in complexity with increasing automation, flexibility and degree of computerisation. This also implies increased requirements for production planning. For many companies modelling and simulation together with a combined optimisation is a strategy to fulfil these requirements. Because of the increasing production planning requirements modelling and simulation environments have to meet these increasing needs.

1.2 Rational for Simulation based Optimisation

Successful systems have been stable over a long time, solved real problems and demonstrated return-on-investment (ROI). New, identical copies of such systems are not risky because they are proved. However, it is not possible to guarantee that innovative system changes will ever generate their ROI. Simulation enables system analysis with time and space compression, provides a robust validation mechanism under realistic conditions and can reduce the risk of implementing new systems. Validation is achieved using a series of qualitative and quantitative experiments with changes of system variables and structures. Pilot projects using real systems with reduced size and/or implemented in a low-risk laboratory environment, can provide analysis results. Such real experiments take time and cost. Hence, a large number of alternatives imply an initial pre-selection. Modelling and simulation can lower the number of alternatives analysed in real experiments as the final step [8].

One reason for system changes is the search for a better overall performance. Under the focus of simulation this means the search for a set of model specifications e.g. input parameters and/or structural assumptions, that leads to an optimal model performance. For all possible variants the range of parameter values and the number of parameter combinations may be too large to implement and simulate manually. A method to automate this is needed. The example described in chapter 6 demonstrates this problem. Even though only a fraction of the complete manufacturing system is modelled the number of possible variants is overwhelming.

Many real word systems are too complex to be expressed by mathematical models. But mathematical models are a precondition of optimisation methods. This leads to a contradiction [2]:

- Pure optimisation models are not able to handle the complexity of both system behaviour and structure.

- Pure simulation cannot find an optimal solution.

⇒ Simulation based optimisation resolves this contradiction through a combination of both methods.

Research and application of simulation based optimisation has seen a significant development in recent years. A Google search on 'Simulation Optimisation' in 2006 found ca. 4.000 entries [2] in comparison to a search in 2008 with almost 80.000 entries among others articles, conference presentations, books and software.

Until a relative short time ago, the simulation community was resistant to the use of optimisation tools. Optimisation models seem to over-simplify the real problem and it was not always clear why a certain solution was the best [8]. The situation changed at the end of the 90s. An ACM Digital Library [57] search on 'Simulation Optimization' found 16.000 articles between 1960 and 2008. A significant number (15.500) of articles has been published during the last 20 years and only 500 articles in the 28 years before. Two reasons

for this change may be the advances in modelling and simulation methods and increase of computing power over the last two decades that has enabled simulation based optimisation.

Currently there are several algorithms to change simulation model parameters to establish solutions with good performance and methods to compare different solutions in terms of quality. Many commercially available discrete event or Monte Carlo simulation software packages contain optimisation methods to search for optimal input and system parameter values [3] e.g. WITNESS with the optional optimisation packages WITNESS Optimizer, ARENA with the additional package OptQuest for Arena [7], SIMPROCESS and SIMUL8 with OptQuest optimisation technology [8].

1.2.1 A Context for Simulation in Manufacturing Systems

The application of manufacturing simulation focuses on modelling the behaviour and the structure of manufacturing organisations, processes and systems. Simulation in a manufacturing system can be used at different phases of manufacturing system lifetime and at different system levels as depicted in figure 1.1. Traditionally, simulation has been used in the planning and design phase dating back to the beginning of the 1960's [26]. Today simulation models are used in all phases of life cycle and at all system levels (see figure 1.1) [19]. Recent developments indicate approaches that also use simulation as an integral part of real time machine control [23] [24] [28].

Figure 1.1 Modelling and simulation of Manufacturing Systems (source [19])

A broad variety of simulation tools are available for manufacturing systems. Historically they can be classified into two major types: simulation languages and application-oriented simulators [26]. Simulation languages are very general. Models are created by coding their behaviour and structure and are similar to a general computer language. Simulation languages provide very high flexibility in model creation but are complex in use for non-scientists and non-engineers. Application-oriented simulators specialise in a given application class. Models are often developed with a graphical user interface based on components, dialog boxes, context menus etc. This eases model development for non-technical users but could lead to reduced flexibility for specific problems [26]. Recent developments indicate that both types are adapting typical characteristics of the other e.g. a simulation language can use a graphical modelling user interface to internally produce code which can be manually altered later.

In summary it is possible to differentiate between general purpose and application-oriented simulation packages. The first are general packages but may have special features for certain application. Examples of general-purpose simulation packages are Arena,

AweSim, Extend, GPSS/H, Micro Saint, MODSIM III, SIMPLE++, SIMUL8, SLX and Taylor Enterprise Dynamics Developer. Examples of application-oriented simulation packages for manufacturing are Arena Packaging Edition, AutoMod, AutoSched, Extend + MFG, ProModel, QUEST, Taylor Enterprise Dynamics Logistics Suite and WITNESS. Short overviews about the above packages and their main feature can be found e.g. in [7] [25] [26].

Other classifications of simulation packages exist, e.g. the differentiation between continuous and discrete simulation. Few systems are completely discrete or continuous but in many systems one is dominant or analysis objectives require the use of a specific simulation type. Due to the stochastic nature of systems continuous processes can be approximated by stochastic distributions with start and stop events. Hence, a continuous system or sub system can be described by a discrete event system. For example, in an automobile assembly line simulation discrete events dominate but of course it would be possible to continuously describe sub systems e.g. work piece movements. In contrast in a chemical plant continuous state changes prevail but the switch of a valve could be modelled discretely.

In this research a general, theoretical established, discrete modelling and simulation approach is used. Hence the research results are general statements and applicable to generic simulation approaches and application specific systems respectively. The Discrete Event System Specification (DEVS), used in this research, is a formalism based on discrete event models. It supports a modular, hierarchical model construction and claimed to be a general and powerful approach in the field of discrete event simulation. The formalism can describe models with a formal specification and simulation model execution with generic simulation algorithms.

1.2.2 Aims and Objectives

The research addresses a fundamental problem of simulation based optimisation. The technique is well established but is restricted to the optimisation of system parameters. In

using these established techniques model structure is considered to be fixed as the structure of model elements is defined during model development before an optimisation experiment. As model performance is optimised it may be necessary to redesign the model structure. This would conventionally be done manually by an analyst using previous simulation results, observations or decisions based on previous experience. This manual process cannot guarantee the global optimal solution. The aim of this research is to develop an approach to discard the manual changes i.e. to develop a combined, simulation based parameter and structure optimisation.

The objectives are:

- Carry out a literature analysis on simulation based optimisation and search methods

- Carry out a literature analysis on the specification and simulation of modular, hierarchical discrete events systems, particularly the Discrete Event System Specification (DEVS) and DEVS extensions

- Advance the established approach of a simulation based parameter optimisation to a simulation based parameter and structure optimisation

- Develop a modelling and simulation method based on DEVS and DEVS extensions to create a merging formalism which combines advantages of different approaches

- Investigate model management and model generation methods

- Investigate appropriate optimisation and search algorithms

- Validate the research and developed approach using an industrial application

- Publish the results in peer reviewed journals, at conferences or in other research publications

1.2.3 Cost Reduction with the Aid of Simulation based Optimisation

The results of this research enable two different possibilities for cost reduction:

1. With increasingly complex, flexible and reconfigurable manufacturing systems the number of possible structure variants increases. In using established approaches it

may be necessary to redesign the model structure between two parameter optimisation runs, normally carried out manually by an analyst using previous simulation results, observations or decisions based on previous experience. This is time consuming and potentially error prone. With this new approach providing automatic reconfiguration and optimisation of both model structure and model parameters the process becomes shorter and the ability to find an optimal solution increases.

2. Many manufacturing systems have the potential to be optimised. Using existing machines, facilities and processes, optimisation could be used to find a new layout and system dimension with improved performance.

The application of this research described in the thesis demonstrates both aspects.

1.3 Methodology and Structure of the Research

The four main areas investigated in this research are:

1. Introduction of *simulation based optimisation* approaches with regard to an extension to a structure optimisation method

2. *Modelling and simulation* method based on the Discrete Event System Specification (DEVS)

3. *Model management and model generation method* using the System Entity Structure/Model Base (SES/MB) framework

4. *Employing* the approach with a real life manufacturing problem

A new approach was established based on the methods 1, 2 and 3. Through the linking of the methods and the definition of appropriate interfaces between them they constitute a new approach to a combined and automatic simulation based parameter and structure optimisation. Figure 1.2 depicts the connections between the investigated areas.
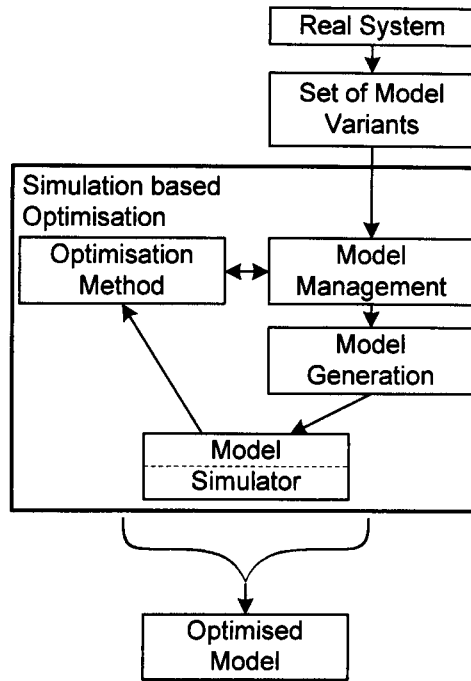
Figure 1.2 Research area structure

1.3.1 Simulation based Optimisation

Modelling and simulation with integrated parameter optimisation to improve model performance is an established technique. In using these established approaches model structure is considered to be fixed as the relationships between model elements (machines, facilities, conveyors etc.) are defined during model development before the optimisation experiment. As model performance is optimised it may be necessary to redesign the model structure after the optimisation experiment. This is normally carried out manually and repeatedly by an analyst with subsequent optimisation experiments.

In established parameter optimisation methods the number of parameters and their domains specify the search space. Depending on the optimisation method the search space is traversed i.e. the optimisation method needs a specific knowledge about the search space bounds. Certain points of the search space are analysed. Each point defines a certain parameter value set. The model is initialised with this parameter value set and subsequently simulated.

The extension using a structure changing facility means broadening the technique to a parameter and structure optimisation. Additional variables with their associated domains are describing possible model structure variants. The combination with the set of parameters defines the new search space of the extended optimisation problem. Methods to transform the set of parameters and structures to a search space definition and vice versa a search space point to a model structure and model parameter values are an integral part of the broadened technique.

1.3.2 Modelling and Simulation

Many different concepts and methods of modelling and simulation exist. This research is restricted to the discrete event system specification formalism, characterised by continuous time and discrete state changes and modular, hierarchical modelling and simulation. The investigated und further developed discrete event system approach is based on DEVS introduced by Zeigler [66] [67] [68]. This approach is one of the most developed, theoretical well-founded discrete event approaches. DEVS supports the definition of modular, hierarchical systems and incorporates well-defined simulator algorithms.

A crucial part of the research is the analysis of the discrete event system specification and the existing extensions with regard to simulation based parameter and structure optimisation and its application in a prototype implementation. Based on the Classic DEVS formalism [66] a broad range of publications with several extending approaches are available. For the application of this research within the manufacturing systems domain certain Classic DEVS extensions were incorporated to establish the Extended Dynamic Structure Discrete Event System specification formalism (EDSDEVS). Consequently a formal concept for this unified specification was developed. The formalism was verified with examples from [66], a benchmark application [18] and industrial applications [16] [17].

This research is a key element of a major search project of the Research Group of Computational Engineering (RG CEA), Hochschule Wismar University of Applied Sciences Technology, Business and Design[1].

1.3.3 Model Management and Model Generation

In a further crucial area of the research the following key features of a model management as part of a simulation based structure optimisation were developed:

- Declarative specification of different model structures
- Definition of a method for external controlled model structure selection
- Definition of an interface between model selection and model generation

To specify a set of modular, hierarchical models an approach has to be able to describe three relationships: (i) *decomposition*, (ii) *taxonomy* and (iii) *coupling* [52] [66] [69].

(i) *Decomposition* means the approach has to be able to decompose a system called entity into sub-entities.

(ii) *Taxonomy* means the ability to represent several, possible variants of an entity called specialisations.

(iii) To compose an entity from sub-entities these have to be connected. This is the meaning of a *coupling* relationship.

The System Entity Structure/Model Base (SES/MB) approach is able to describe these three relationships [52], [66], [69]. The original SES/MB approach was developed to assist a manual model design process for modular, hierarchical models using a tree like definition with different node and edge types and a model base containing basic components. An essential demand for an appropriate model management method is the external controllability. The SES/MB approach has to be changed to comply with this demand.

Based on the adapted SES/MB approach three interfaces around the model management method were designed. The first interface is a model set definition based on a

---

[1] Research Group Computational Engineering and Automation, http://www.mb.hs-wismar.de/cea/

XML file structure. This interface is deployed to create a specific SES/MB structure. In future extensions the development of a graphical SES/MB modeller based on this interface would be possible. The second interface delivers model generation information to a model generator. It is based on a XML file structure definition. This interface represents the connector to the modelling and simulation method. The third interface communicates with the optimisation methods during the initialisation and the optimisation phases:

1. In the initialisation phase it delivers information about the search space defined by the set of all possible model structure and model parameter variants to the optimisation method.

2. During the optimisation phase it receives information from the optimisation method about the currently investigated search space point. This information is used to select the corresponding model structure and initialises the model parameters. A subsequent model structure validation is a crucial part of the model structure selection.

1.3.4 Implementation and Employment

In this research methods and algorithms were implemented using the MATLAB Scientific Computing Environment [58].

1. The modelling and simulation toolbox was not started from scratch. A pre-release of the modeller and simulator published in [41] was the starting point. These sources were adapted to the current MATLAB version with a new object-oriented programming principle and were extended step-by-step. Each extension was validated with test models for example those introduced in [66]. Each important stage of the research was published and subject to peer review [16] [17] [18] [34].

   A simulation model was implemented as a basis for later optimisation. This model uses results, observations, structures, parameter etc. gathered by the author of this thesis during several projects which were realised by the supporting company

Syntax Software[2]. The company is a leading production and machine control software developer for the photofinishing industry. The final model was validated with original production data taken from photofinishing applications implemented by the author.

2. The model management toolbox was developed and tested using conventional software engineering techniques.

3. The optimisation method used the commercial available Genetic Algorithm Toolbox [59].

4. The research application is based on industrial experience of the author. The germ of the idea to optimise structure comes from a project enquiry made by the Kodak Photofinishing Department to Syntax Software 6 years ago. The project was not realised because Kodak closed their European photofinishing business.

To validate the new approach all possible model variants were simulated. The simulation results are compared with the result of the automatic structure and parameter optimisation. This procedure and its results are described and discussed in chapter 6.

1.4 Research Outcomes

The outcomes of this research can be divided into four parts:

1. Development of an approach for a combined, simulation based model parameter and model structure optimisation

   The extension of the established simulation based parameter optimisation by a controllable model management is the fundamental idea behind this research. Through this inclusion of a model management the optimisation method can simultaneously control parameter changes as well as model structure changes to find an optimal system configuration.

2. Development of an Extended Dynamic Structure DEVS Formalism

---

[2] SyntaX Software Inh. Jörn Satow formerly SyntaX Software O.Hagendorf J.Satow GbR, Schweinsbrücke 9, 23966 Wismar, www.syntaxsoft.de

Classic DEVS and DEVS extensions has been a research topic since more than 30 years. The extensions have one joint attribute: they are based on the Classic DEVS formalism. Hence, the decision on one DEVS extension inhibits the use of advantages of another one. In this research selected extensions are combined to create to a merging formalism to combine the advantages of different approaches.

3. Validation of the new approach

The approach was successfully validated with a simulation based optimisation experiment using an industrial application. All variants of the application were calculated and the results compared with the optimisation experiment. The global optimal result was found with a probability of 47%. With an error of 3% of the system performance an optimal result was found with a probability of 68%. To find an optimal result, on an average 70% of the search space were analysed. With a second experiment the dependency of optimisation results on search method configuration was shown. However, the finding of an optimal search method configuration was not within the scope of this research.

4. Publication of results

Results and intermediate steps have been published in a peer-reviewed journal and as a book chapter and have been presented at international conferences.

1.5 Contribution to Knowledge

This research has resulted in two novel formalisms:

1. an approach to extend the established simulation based parameter optimisation to a combined simulation based parameter and structure optimisation which automatically change system structure and parameter values to improve the overall system performance

2. an Extended Dynamic Structure Discrete Event System Specification (EDSDEVS) as an enhancement and combination of the Discrete Event System Specification and

some of its different extensions. The EDSDEVS formalism is used as one component of the simulation based parameter and structure optimisation approach. The contribution and the advantages of this approach are:

- The approach establishes a structure and parameter optimised model based on the definition of a set of model variants. The previous manual steps of changing structure to find an optimal system model are now incorporated into an optimisation algorithm and thus are automated.

- Through automation the probability of finding the optimal solution grows significantly in comparison to a manual search.

The contribution and the advantages of the EDSDEVS approach are summarised as follows:

- fusion of different extensions of the Classic Discrete Event System Specification

- implementation of modelling and simulation environment for research and teaching

1.6 Contents of this Thesis

The thesis is organised into three main sections as depicted in figure 1.3. In chapter 2 the simulation based optimisation is introduced, limitations are outlined and the idea of an extension of the established technique is developed. Based on this new concept of a simulation based parameter and structure optimisation the requirements of several algorithms, methods and interfaces are brought out. Essential components of the optimisation concept are appropriate model management and modelling and simulation methods.

Chapter 3 starts with a short presentation of simulation and simulation model taxonomy. The Classic DEVS formalism with the associated formal modelling concept and simulation algorithms is introduced. Concepts of selected extensions of the DEVS formalism are subsequently shown. The last part of chapter 3 introduces the EDSDEVS formalism as it was developed in the scope of this research. The formal concept of EDSDEVS, the dynamic behaviour of its components in different situation and simulation algorithms are shown.

Chapter 4 introduces the System Entity Structure/Model Base framework as an approach to organise a set of model structure variants based on meta-modelling. In chapter 5 all aspects of this approach for a simulation based parameter and structure optimisation are described in detail.
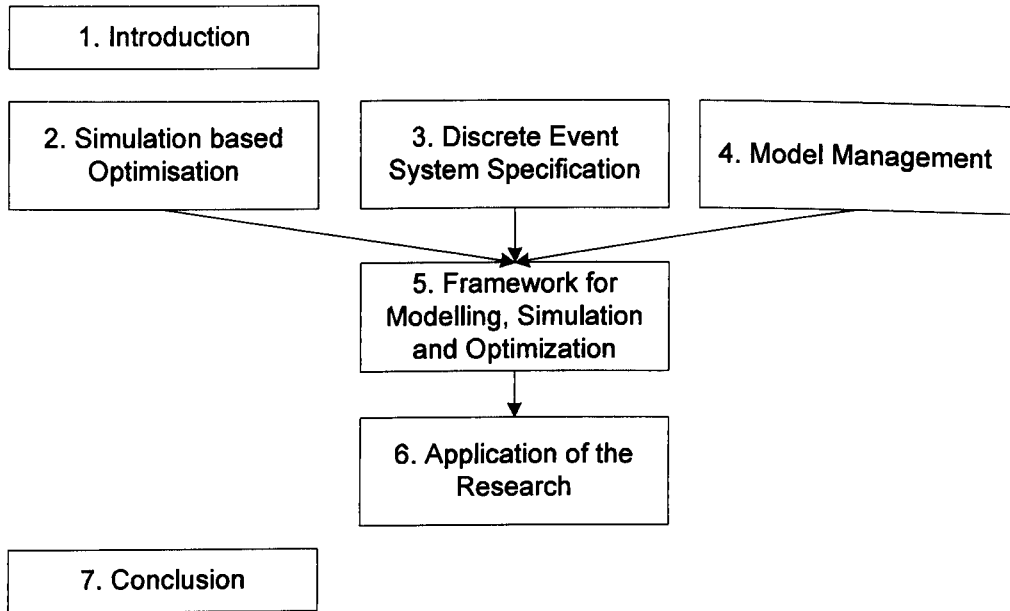
Figure 1.3 Structure of the main sections of the thesis

Chapter 6 demonstrates application of the approach with an optimisation example. The problem is taken from the industrial experience of the author. The general structure of a photofinishing lab i.e. a company for industrial production of photos and related products is described together with a daily problem and how this could be solved with the new approach of a simulation based optimisation.

The thesis concludes with a summary and suggestions for further work.

Chapter 2

Simulation based Optimisation

Optimisation is an important research topic and has the potential for significant commercial application. At the ACM Digital Library [57] the first publications on optimisation were published in the early 1950s, ca. 118.000 to date. They cover a very broad range of optimisation methods and optimisation applications. In general, the aim of an optimisation method is to find an optimal problem solution in a given search space whereas the often multidimensional search space defines the complete set of possible problem solutions.

Research and application of simulation based optimisation has seen a significant development in recent years. A Google search on 'Simulation Optimisation' in 2006 found over 4.000 entries [2] in comparison a search in 2008 found almost 80.000 entries among others articles, conference presentations, books and software.

The integration of optimisation techniques into simulation packages has been an important requirement for commercial modelling and simulation tools, shown for example in comparing two popular simulation textbooks [7] and [25] with previous editions. The third edition of Law and Kelton [25], published in 2000, lists five commercial available simulation based optimisation tools which did not exist at the time of the second edition of the book, published 1991 [15].

The following chapter introduces the ideas of combining modelling and simulation with optimisation methods. It concludes with the introduction of the new simulation based parameter and structure optimisation approach developed in this research.

2.1 Introduction

In retrospect a disadvantage of modelling and simulation is the missing optimisation capability. For many years, simulation experiments as shown in figure 2.1 have been state of the art. An analyst creates a model e.g. based on a real system, transforms the model to an executable model and executes a simulation with it. After a review of simulation results the model configuration, i.e. model parameters and/or model structures has to be manually changed by an analyst, when necessary. Using a manual procedure only a relative small number of system configurations can be examined until a suitable solution is chosen. It is not possible to guarantee the detection of an optimal or near optimal system configuration and the manual effort to find a solution can be considerable.
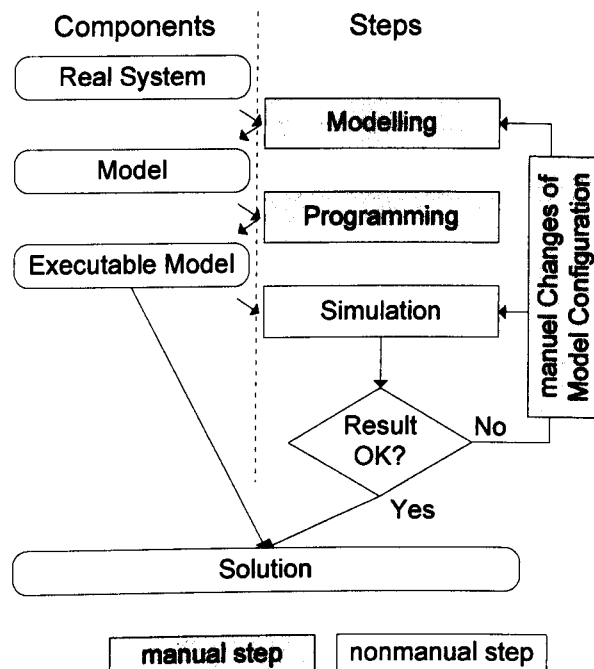
Figure 2.1 An example of an conventional simulation experiment

Through the combination of modelling and simulation with optimisation methods to a simulation based optimisation method this manual procedure can be partly automated. Mathematical optimisation generally means establishing a function minima or maxima. Simulation based optimisation means finding the best model configuration by minimising a

function of output variables estimated with a simulation method [56]. Important prerequisites are the availability of:

- suitable modelling and simulation methods

  Modelling and simulation as well as model and model parameter have to be strictly separated. With the combination of optimisation and simulation an optimisation method needs capabilities to influence the model configuration.

- suitable optimisation methods

  Figure 2.2 shows a classification of optimisation methods, identified during this research, many others and more completed classifications exists in the optimisation literature. Enumerating or calculus based optimisation methods are suitable when the search space is small enough and the problem is analytically solvable respectively. If the problem complexity is large, often search based algorithms are more appropriate. Problem descriptions with a stochastic component are another crucial reason to use a search based optimisation method. Because of the typical stochastic character of a simulation calculus based optimisation methods are not appropriate for a simulation based optimisation.

- sufficient computing power

  Simulation based optimisation is typically used when the number of different model configurations is large. This is often accompanied with complex model structures. Both results in considerable quantity of computing time while searching for the optimal model configuration.

Descriptions of established and new simulation based optimisation approaches follow in sections 2.2 and 2.3.
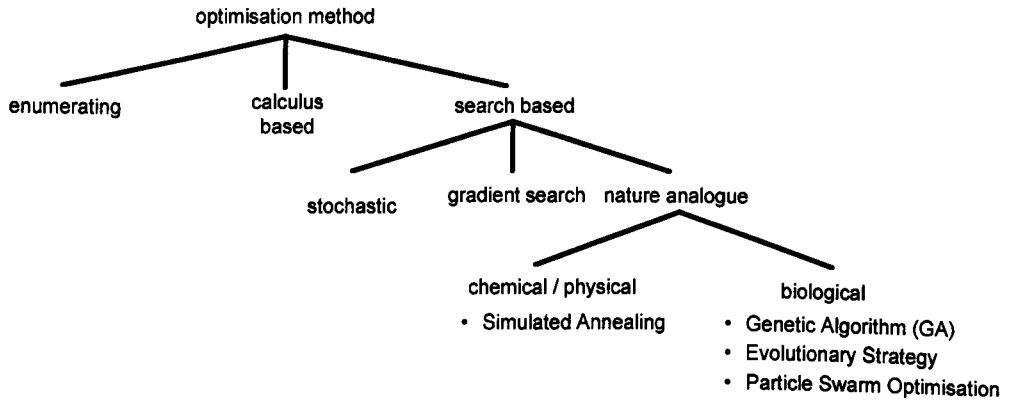
Figure 2.2 Classification of optimisation methods

2.2 Parameter Optimisation

An established approach to simulation based optimisation is simulation based parameter optimisation. The overall goal of this optimisation approach is the identification of improved settings of user selected model parameters under control of performance measures. There is a extensive and varied body of literature on this topic that includes several tutorials, reviews and summaries of the current state of the art (e.g. [4], [6], [14], [32], [55], [56]). Law and Kelton describe in [25] commercial available simulation tools with integrated optimisation techniques using this approach of simulation based parameter optimisation. Figure 2.3 shows a principle example of a simulation based parameter optimisation experiment. The procedure to create an executable model follows the procedure described in figure 2.1. A crucial difference is the detachment of model and model parameters. Based on this detachment the optimisation method is able to alter the model parameter set to improve the result of an objective function. The objective function measures the model performance with current model parameters i.e. improving the objective function result means improving the model performance. Model parameter adjustments are carried out in a loop until a stop criteria is fulfilled. Examples of stop criteria are (i) going below a minimum alteration rate or (ii) exceeding the maximum number of optimisation cycles. The result of a successful optimisation experiment (example criterion (i) fulfilled) is a parameter optimised model.
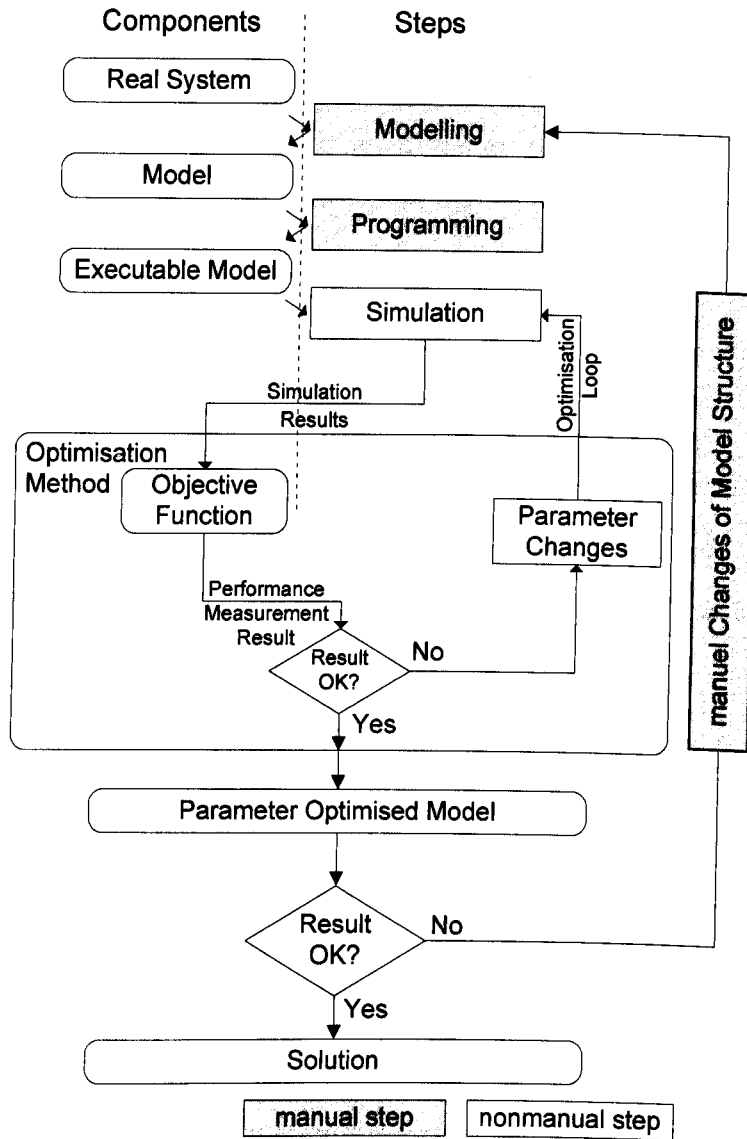
Figure 2.3 An example of a simulation based parameter optimisation experiment

According to [56], a simulation based parameter optimisation problem O with a set of m deterministic model parameters $X = \{x_1, \ldots x_m\}$ can be formally described as follows:

- A parameter set $X = \{x_1, \ldots x_m\}$ has the domain set $D = \{d_1 \ldots d_m\}$

- The multidimensional (one for each parameter) search space S is defined by

  $S = \{s = \{v_1 \ldots v_m\} \mid v_i \in d_i\}$

- A set Y is the output set defined by $Y = \{y_1 \ldots y_n\} = Y(X)$ and estimated by simulation. Simulation experiments are often based on stochastic model properties.

Hence the output set Y is stochastic.

- The objective function F establishes a single stochastic value from stochastic output set Y : $F = F(Y(X)) \rightarrow \mathfrak{R}+$. The result of the objective function is a measure of the current model performance.

- Because of the stochastic nature of Y and consequently of F, an estimation function R, the simulation response function defined by $R(X)=E(F(Y(X)))$, is optimised, i.e. in the scope of this approach it is minimised.

- Depending on optimisation problem and analysis required the exchange of the last two steps, evaluation of objective function F and simulation response function R, can save computational effort. Hence, the simulation response function is defined by $R(X) = E(Y(X))$ and subsequently the objective function by $F(X) = F(R(X))$.

Each parameter set $X_i \in S$ can be seen as a possible solution of O. The optimisation method has to search the search space S to find the parameter set $X_{opt} \in S$ with $E(F(Y(X_{opt}))) \leq E(F(Y(X_i))) \ \forall X_i \in S$. The resulting parameter set X_{opt} is considered the global optimum of O.

This approach is restricted to automated parameter optimisation. It is important to note that automatic structure changes during optimisation are not possible with this approach. Instead, structure changes are carried out manually by an analyst and each manual structure change requires a repetition of the automated parameter optimisation.

2.3 Parameter and Structure Optimisation

The extension of the optimisation approach with the ability to also change model structures to improve system performance is a development of the idea introduced in section 2.2. This extension is mainly directed towards a simulation based structure and parameter optimisation as presented in figure 2.4. The approach of a simulation based parameter and structure optimisation differs in the following extensions or modifications from the simulation based parameter optimisation depicted in figure 2.3:

- An analyst does not generate a single model of the real system. In this case he has to organise a set of models. One way of achieving this is to define a model that describes a set of model variants instead of one single model of the system under analysis. Models that define the creation and interpretation of a set of models are named meta-models. If a model is the abstraction of an aspect of the real world, a meta-model is yet another, super-ordinate abstraction of the model itself. That is when a model describes the behaviour and structure of a real system then a meta-model describes the behaviour and structure of different models that all describe the behaviour and structure of the same real system in a slightly different way.

- The model management organises the set of model structures and provides a model selection method.

- The model selection is controlled by a superior optimisation. The selection method delivers the selected model structure information to a model generator which generates an executable model. The parameter transfer and the simulation match the simulation based parameter optimisation depicted in figure 2.4.

- The objective function receives simulation results to estimate the performance of current model structure and parameters similar to the approach depicted in figure 2.4. Information generated by the model selection method can be additionally used to establish the model performance.

- The optimisation method investigates the search space with simultaneous model parameter and model structure changes without a manual involvement. The intention of the optimisation method is the finding of a model structure and model parameter set where the objective function delivers the global optimum value, in most instances the global minimum.

Components   Steps
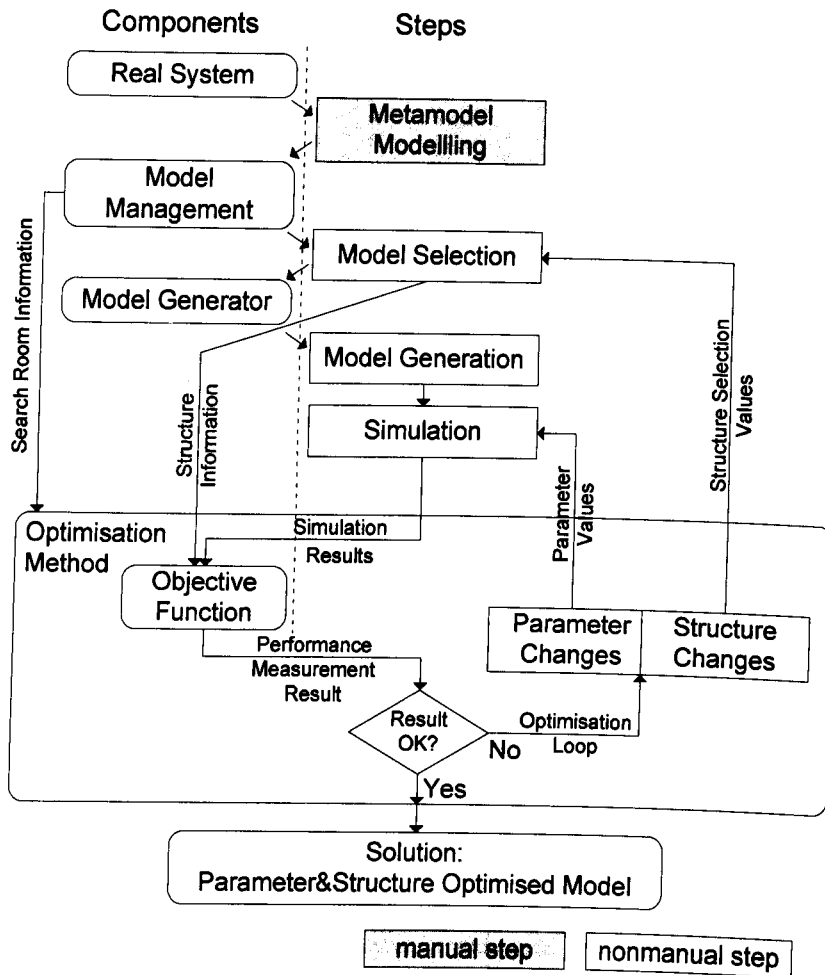
Figure 2.4 Components and steps of a simulation based parameter and structure optimisation experiment

A prerequisite for an optimisation is the definition of a search space. In the approach presented here, the search space is multi-dimensional as a result of the combination of model structure and model parameter variants. During the optimisation loop several points of the search space are examined. Each point defines a model structure with an appropriate parameter set. The extension of the formal description of a simulation based parameter optimisation problem O, defined in section 2.2, to a combined simulation based structure and parameter optimisation leads to O^*:

- The model parameter set X_P and its domain set D_P, in section 2.2 defined as X and D, are extended by structure parameter set X_S and its domain set D_S. The extended set

definitions are: $X^* = X_P \cup X_S = \{x_{P1} \ldots x_{Pm}, x_{S1} \ldots x_{Sn}\}$ and

$D^* = D_P \cup D_S = \{d_{P1} \ldots d_{Pm}, d_{S1} \ldots d_{Sn}\}$ with m model parameters in set X_P and

n structure parameters in set X_S. The sets X_P and D_P are defined by the current model.

The model management has to provide the sets X_S and D_S by analysing the meta-

model.

- The multi-dimensional (one for each parameter) search space $S = S_P \cup S_S$ is spanned

  by sets of model parameter and structure variants.

- The objective function F^* is defined by $F^*(Y(X^*), P(X_S))$ with simulation results

  $Y(X^*)=Y(X_S \cup X_P)$ and results based on structure related variables $P(X_S)$ which are

  established during the model selection. Because of the stochastic nature of the

  simulation results $Y(X^*)$ an estimation function R, the simulation response function,

  is calculated. The results based on structure related variables $P(X_S)$ are not

  stochastic. Hence, the simulation response function is defined by $R(Y(X^*))$ and

  subsequently the objective function by $F^*(R(Y(X^*)), P(X_S))$.

Figure 2.5 depicts the above formal description of a simulation based parameter and

structure optimisation framework O^* in a schematic diagram.
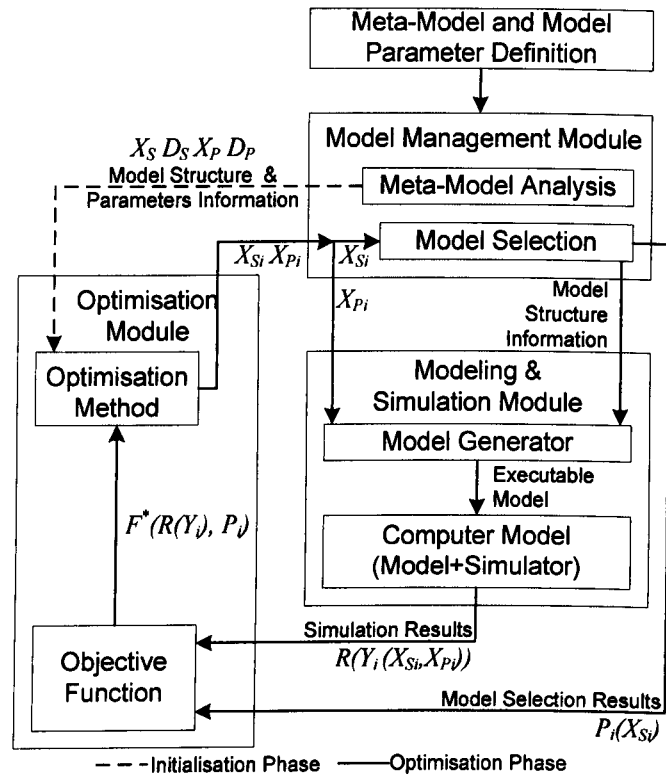
Figure 2.5 Schematic diagram of a simulation based parameter and structure optimisation

framework

Further prerequisites of the introduced approach are:

- The modelling and simulation method with support of modular or modular, hierarchical models and a flexible simulation engine are essential parts of the framework. A powerful modelling and simulation method is fundamental in two different aspects: (i) A strict separation between model and simulator are necessary due to the crucial management of a model structure set with a downstream model generator and a model parameter transfer. (ii) A flexible and modular, hierarchical modelling and simulation method can incredible enlarge the application field and ease its use.

- The cooperation between optimisation, model management, and modelling and simulation modules has to be comprehensive. The aim of the cooperation is to establish control of both model parameters and model structures by an optimisation

[27]

method. The objective function evaluates simulation results but can also incorporate further information, generated by model management, into the evaluation. The additional parameters can be provided by optional variables, summarised during model selection as described in section 4.2. The search space definition used by the optimisation module is established by the model management module. These information exchanges require comprehensive cooperation between the above modules.

- Using combined simulation based structure and parameter optimisation the number of variants of different system configurations can be considerable higher than in a pure simulation based parameter optimisation and will need more computing power than the approach described in section 2.2.

Through the inclusion of a model management method, the optimisation method can simultaneously control parameter changes as well as model structure changes to find an optimal system configuration. This new approach significantly enhances the application of simulation based optimisation. The extension of the simulation based parameter optimisation by a controllable model management and subsequent automatic model generation is a fundamental idea behind this research.

The modelling and simulation and model management methods take a crucial role in this approach. The description of a discrete event modelling and simulation method, and a model management method based on meta-modelling follow in the next two chapters.

Chapter 3

Discrete Event System Specification and Simulation

After a short, general introduction to modelling and simulation this chapter explains the DEVS formalism. The Classic DEVS formalism will be introduced together with several extensions which are combined to form an Extended Dynamic Structure DEVS (EDSDEVS) approach. The chapter concludes with the introduction of the EDSDEVS formalism. The EDSDEVS modelling and simulation approach with its advanced, modular, hierarchical model definitions and flexible simulation algorithms plays a major role in the new simulation based optimisation approach.

3.1 Introduction

A simulation is the imitation of the behaviour and the structure of a real-world system. The behaviour and the structure of the system are studied by developing a simulation model and performing experiments with it. During an experiment the model is executed within a simulation environment by a simulator. The model is usually created by taking assumptions concerning the function of the system, its attributes and structures. The complete system is split into several entities with relationships defining connections between them. A more complex system can be split in a hierarchical manner i.e. an entity can be segmented into sub-entities which themselves can be again segmented into sub-entities. The entities are expressed in a mathematical, logical or symbolic form. Once developed and validated a model can be used to perform a variety of analysis concerning the real-world process or

system. Analysing experiments can change the behaviour or the attributes of a certain entity, the relationship between entities or sending changed inputs to the model.

It is possible to summarise as follows and as shown in figure 3.1:

- Modelling and simulation is the imitation of a real-world system.

- The model tries to describe real-world behaviour through states, state-transitions and attributes.

- The model tries to describe the real-world structure throughout partitioning into sub-entities. Subject to the modelling formalism, the structure can be defined hierarchically.

- The model interacts with its environment based on inputs and outputs.

Figure 3.1 A real-world process or system and its model (source [1])

Under some circumstances, a model can be developed based on mathematical methods only e.g. by the use of differential equations, algebraic methods or other mathematical techniques. However, many real world systems are to complex to be modelled using mathematical expressions. In these cases, numerical, computer based modelling and simulation can be used to analyse the behaviour and the structure of real word systems [7].

Many different concepts and methods for modelling and simulation exist. Ören [33] classifies different types of simulation models with several criteria. One of the various possible classifications is to use the two criteria - time change and state change [48]. Discrete event models are a combination of continuous time and discrete state changes as shown in figure 3.2. The choice of whether to use discrete state changes, continuous state changes or a combination of both depends on the characteristics of the system under investigation and the objectives of the study.

Figure 3.2 Simulation model taxonomy (source [48])

The Discrete Event System Specification (DEVS) is a formalism based on discrete event models. It supports a modular, hierarchical model construction and claimed to be a general and powerful approach in the field of discrete event simulation [66] [67].

For modelling and simulation and particularly with DEVS the term formalism is used with a specific meaning. A modelling formalism can be described by two parts: (i) formal model specification and (ii) simulation algorithms to execute the model [53]. The

formal mathematical specification describes model structure and behaviour. The simulation algorithms specify methods to execute any model that is described in accordance with the formal model specification.

3.2 Discrete Event System Specification

The DEVS formalism was first introduced by Zeigler [68] in the 1970s. In [66] the authors classify this formalism, position and compare it with other, more established modelling and simulation formalisms. Several international research groups are working on the DEVS formalism and are regularly publishing results at the annual DEVS Symposium at Spring Simulation Conferences. Wainer [62] maintains a list of available DEVS tools. The DEVS formalism is, in contrast to other modelling and simulation formalisms, not very widely used in industrial practice. This situation exists despite the fact that the theory is a well-founded, general formalism. It can only be assumed that one reason of the marginal acceptance is the type of available software tools [34].

Since its first publications, in [68] the formalism has been enhanced and many extensions have been introduced. To differentiate among them the original formalism is termed Classic DEVS.

3.2.1 Classic DEVS Modelling

DEVS is a modular, hierarchical modelling and simulation formalism. Every DEVS model can be described by using two different model types, atomic and coupled. Both model types have an identical, clearly defined input and output interface. An atomic model describes the behaviour of a non-decomposable entity via input/output events and event driven state transition functions. A coupled model describes the structure of a more complex model through the aggregation of several entities and their couplings. These entities can be atomic models as well as coupled models. Due to the identical interfaces and the complete encapsulation of a model, a coupled model cannot differentiate between the different model

types of its sub components. A coupled model does not need and does not even have any information about the type of its sub-entities. The internal structure of each sub model is completely encapsulated and separated from its parent. Due the possibility that several entities together create a new entity which itself can be again part of another super-ordinate entity the formalism is termed 'closed under coupling'. Thus, the construction of modular, hierarchical models is possible [66].

am   atomic model
CM   COUPLED MODEL
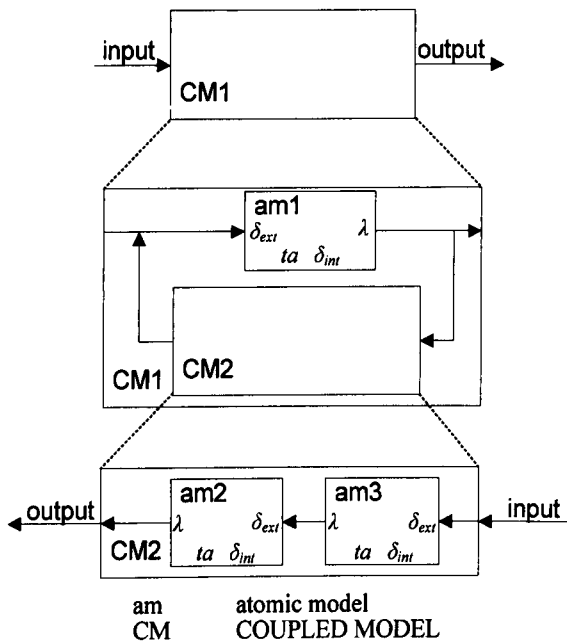
Figure 3.3 DEVS model example

Figure 3.3 shows a DEVS model example:

- Structure description:

  The structure of the real-world system is depicted by the structure of the DEVS model i.e. the aggregation of entities and sub-entities and their directed coupling relations. The top most model i.e. the root model depicts the real-world system with an interface to its environment. This external interface is defined by the input and output ports of the root model. The environment is modelled in an Experimental Frame as described in [11] [66]. An Experimental Frame makes the analysis of the modular, hierarchical model possible, generates input events and analyses the output events. The sub-entities input and output ports are connected over directed couplings

with other sub-entities input and output ports and with the output port of the super-ordinate coupled model, respectively. Each atomic and coupled model has one input and one output port. Depending on source and destination port the coupling relations are named:

- external input coupling (EIC) with the input port of a super-ordinate coupled model as source and one or more sub-entities as destination

- external output coupling (EOC) with the output port of a sub-entity as source and the output port of a super-ordinate coupled model as destination

- internal coupling (IC) with output and input port of sub-entities as source and destination

Example:

The coupled model *CM1* in figure 3.3 is the top most model i.e. the root model. The root model has an external interface with input and output ports to handle or create external input and output events received by or sent to the experimental frame. It contains one atomic model *am1* and one coupled model *CM2*. The coupled model *CM2* consists of two atomic models *am2* and *am3*. As an EIC the input port of *CM1* is connected to the input port of *am1*. As an EOC the output port of *CM1* forwards events sent from the output port of *am1*. ICs are the connections between the output port of *am1* and the input port of *CM2*, output port of *CM2* and the input port of *am1* and output port of *am3* and the input port of *am2*.

- Behaviour description:

The behaviour of a real-world system and sub system, respectively, is depicted by an atomic model and its internal states, input/output events and event driven state transition functions. At its input port it can receive external input events. An input event is handled by an external state transition function. This function can immediately but indirectly induce an internal event and subsequently an internal

transition. With time controlled internal transitions an atomic model can react to time events. Internal events are scheduled by a time advance function and their state transitions are handled by an internal state transition function. After each external and internal event the time advance function is called to schedule the next internal event. With output events send from an output port the atomic model can influence other entities connected to this port or create the output event of the super-ordinate coupled model. Output events are created by an output function which is firstly executed during internal event handling before calling the internal state transition function.

Example:

The atomic model *am1* in figure 3.3 executes the external state transition function δ_{ext} when it receives an input event. After initialisation and after each event handling the next internal event is scheduled with the time advance function *ta*. During the internal event handling by model *am1* the internal state transition function δ_{int} is called. Before the function δ_{int} is called an output event can be created by executing the output function λ.

- Event handling:

All input events are received over the input port regardless of event source and type. All output events are sent over the output port regardless of event type. An event received at an input port of a coupled model is forwarded to the connected sub-entity(s). An event send to an output port of a coupled model by a sub-entity is received and handled by the super-ordinate coupled model. An event send by a sub-entity to one or more sub-entities of the same coupled model is routed by this coupled model from sending output to receiving input port.

Example:

When *CM1* in figure 3.3 receives an event at its input port it is forwarded over the EIC to *am1*. When *CM2* forwards an output event to its output port, the event

is forwarded to the input port of *am1* over the IC. When *am1* generates an output event at its output port this event is forwarded to *CM2* due to an IC and simultaneously it represents an output event of *CM1* due to an EOC.

3.2.2   Formal Concept of Classic DEVS Modelling

The Classic DEVS formal description defines coupled and atomic models as a combination of sets and functions. The description of an atomic model is a 7- tuple [66]:

$AM = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$

- X, Y and S specify the sets of discrete inputs, outputs and internal states.

- δ_{ext}: $Q \times X \rightarrow S$ where $Q = \{(s,e) \mid s \in S, 0 < e < t_{next}$, *elapsed time e = t - t_{last}}*

  The external state transition function δ_{ext} handles external input event at time t. It can induce an internal transition with a rescheduling of the time of the next internal event. The time of the external input event is stored in t_{last}.

- δ_{int}: $S \rightarrow S$

  The internal state transition function δ_{int} can establish a new internal state. The execution of output function λ and internal state transition function δ_{int} is induced by a time driven internal event. The time of an internal event is established by the time advance function ta. The time of the internal event is stored in t_{last}.

- λ: $S \rightarrow Y$

  The output function λ can generate an output event. If and which output event is generated depends on the internal state S.

- ta: $S \rightarrow \Re_0^+ \cup \infty$

  The time advance function ta schedules the time of the next internal event after each state transition.

Figure 3.4 shows the dynamic behaviour of an atomic model. Listing B.1 in appendix B shows a pseudo code skeleton of an atomic model.
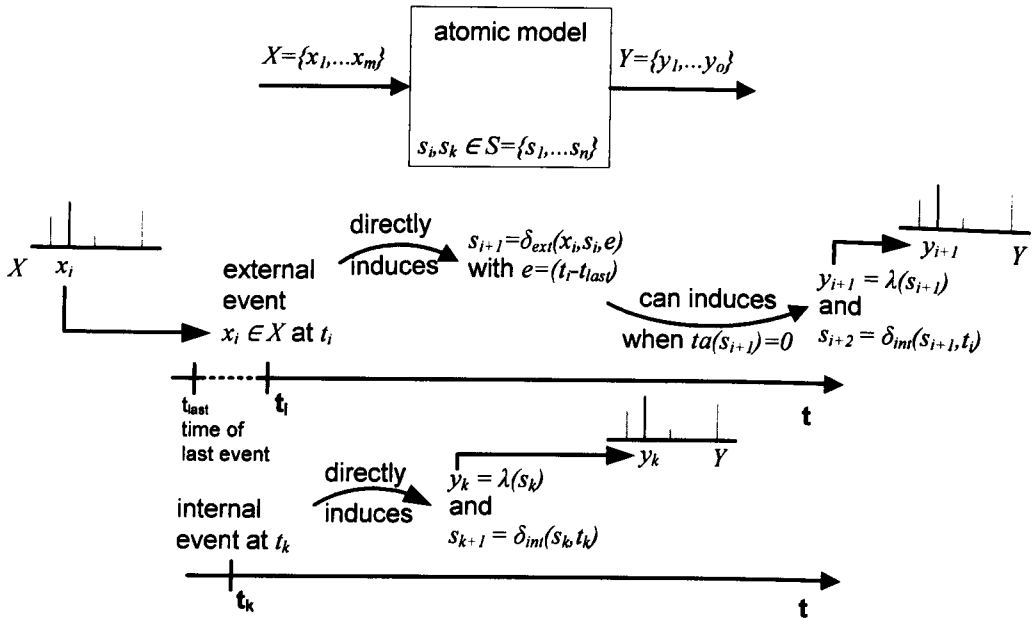
Figure 3.4 Dynamic behaviour of an atomic model

The description of a coupled model is a 9-tuple [66]:

$$CM = (d_n, X, Y, D, \{ M_d \}, EIC, EOC, IC, SELECT)$$

- d_n specifies the name of the coupled model.

- X and Y specify the sets of discrete inputs and outputs.

- D specifies the set of sub component names.

- $\hat{M}_d \mid d \in D$

  M_d is the model of the sub component d

- EIC, EOC and IC are the sets of external input, external output and internal couplings.

- The $SELECT$ function prioritises concurrent internal events of sub components.

The figure 3.5 depicts the relations of the elements of a Classic DEVS coupled model.

Listing B.2 in appendix B shows a pseudo code skeleton of a coupled model.

EIC   = {{CM.X,Comp1.X} {CM.X,Comp2.X}}
IC   = {{Comp1.Y,Comp3.X} {Comp1.Y,Comp4.X}}
EOC   = {{Comp3.Y,CM.Y} {Comp4.Y,CM.Y}}
D   = {"Comp1", "Comp2", "Comp3", "Comp4"}
$\{M_d \mid d \in D\}$ = $\{M_{Comp1}, M_{Comp2}, M_{Comp3}, M_{Comp4}\}$
SELECT   : $priority\_order(M_{Comp1}, M_{Comp2}, M_{Comp3}, M_{Comp4})$
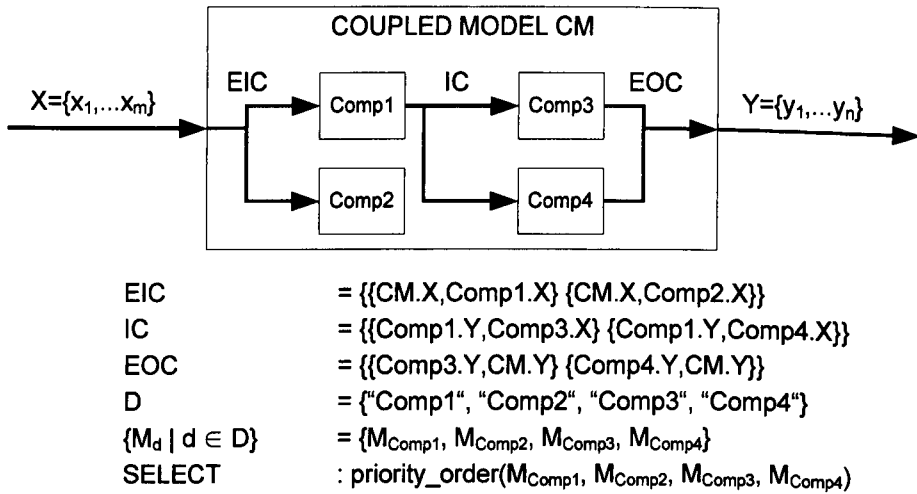
Figure 3.5 Coupled model elements

The Classic DEVS approach supports the specification of behavioural system dynamics in atomic systems and the specification of static component aggregations in coupled systems. It is not possible to describe structural system dynamics at the coupled model level, i.e. the deletion or creation of components and couplings or changes of interfaces, although all necessary structural information is also available during simulation time as is described in section 3.2.3. The only possibility to realise a structural system dynamic is to specify it with logical constructs at the atomic model level. However, this removes the advantages of reusability and model clarity and increases modelling complexity.

3.2.3 Classic DEVS Simulation

Beside the formal definition the second part of the Classic DEVS formalism is the description of abstract simulator algorithms for the execution of DEVS models. The algorithms are named abstract because they are implemented as a general pseudo code. The abstract simulator has a modular, hierarchical structure matching exactly the modular, hierarchical structure of a DEVS model. A DEVS model can be directly transformed into an executable simulator model using abstract simulator elements e.g. as in [48] [66] [67] shown. The abstract simulator approach consists of three different elements namely root coordinator, coordinator and simulator. The structure corresponds to the hierarchical DEVS model

structure except the root coordinator added as the topmost entity. Each atomic model is associated with a simulator element and each coupled model is associated with a coordinator element.

Figure 3.6 shows the transformation of a DEVS model to an executable simulation model using associated abstract simulator elements. The two coupled models *CM1* and *CM2* are mapped to two coordinator elements. The three atomic models *am1...am3* are mapped to simulator elements.
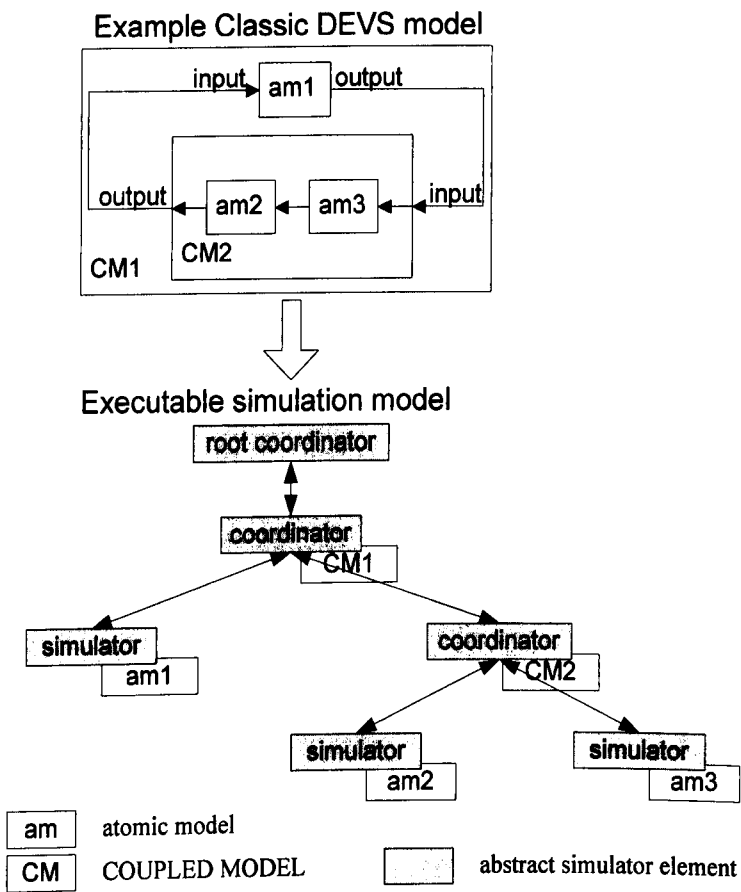
Figure 3.6 An example of a Classic DEVS model with associated abstract simulator elements

The communication between *root coordinator*, *coordinator* and *simulator* instances is message based. On top of the hierarchy the *root coordinator* initiates, controls and ends a simulation cycle with different messages. It holds the simulation clock. Each coupled model is associated to a *coordinator* instance. The *coordinator* instance forwards messages to its subordinated *coordinator* and/or *simulator* instances. It holds the minimum time of the next

internal transition event of its sub components in t_{next}. Each atomic model is associated with a *simulator* instance. It holds the time of its own next internal events in t_{next}. It is important to note that both *coordinator* and *simulator* instances have the same interfaces and receive the same messages. Hence, a super-ordinate coordinator does not have to distinguish the type of subordinate instances.

With this concept one prerequisite of a parameter and structure optimisation approach as introduced in section 2.3 is fulfilled. The modular modelling and flexible simulation play a crucial role in model management and subsequent model generation.

Furthermore this concept enables that the modular hierarchical structure of a model remains an unchanged part of the computational model during simulation runtime. The preservation of the model structure is an essential prerequisite to the dynamic structure modelling and simulation concept introduced later in this chapter. This dynamic structure modelling and simulation concept fulfils another prerequisite of parameter and structure optimisation approach.

Figure 3.7 depicts the structure of a Classic DEVS model with the corresponding abstract simulator instances. Moreover, the figure presents the different messages types passed between the several instances of abstract simulator elements and the subsequent DEVS model function calls. Because of complexity and clarity selected situations are shown in sections:

i. (Figure 3.7a) initialisation phase with i-message handling:

During the initialisation phase model component's init functions are called because of an i-message handling.

ii. (Figure 3.7b) *-message handling created due to internal event of model *am3* with a subsequent x-message within the same coupled model:

The root coordinator advances the simulation clock and a *-message is firstly created. The message is sent to the successor coordinator instance of coupled model *CM1*. This coordinator instance determines that the sub component *CM2* is

responsible for handling this event. Hence, the event is forwarded to the successor coordinator instance of *CM2*. The coordinator instance determines that one of its sub components scheduled the event. The simulator instance of model *am3* initiates the internal message handling. Due to the current internal state of *am3* an output message is generated. With the internal coupling *am2-am3* the message is received as an x-message by simulator instance/model *am2*.

iii. (Figure 3.7c) *-message handling created due to an internal event of model *am1* with a subsequent x-message at different model levels:

The beginning of the message handling is similar to ii except the generated output message is forwarded to another model level over internal and external input couplings.

iv. (Figure 3.7d) *-message handling created due to concurrent internal events of models *am2* and *am3*:

The root coordinator advances the simulation clock and a *-message is firstly created. The message is sent to the successor coordinator instance of coupled model *CM1*. This coordinator instance determines that the sub component *CM2* is responsible for handling this event. Hence, the event is forwarded to the successor coordinator instance of *CM2*. The coordinator instance determines that two sub components scheduled the event. The coordinator instance will then call the select() function to decide which sub components has a higher priority and forward the message to the appropriate simulator instance. The simulator instance calls the model functions λ and δ_{int}. A result of calling λ could be a y-message sent back to the subordinate coodinator instance of *CM2*.
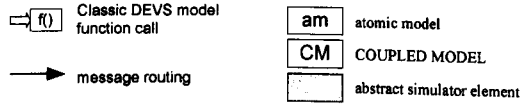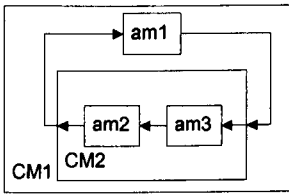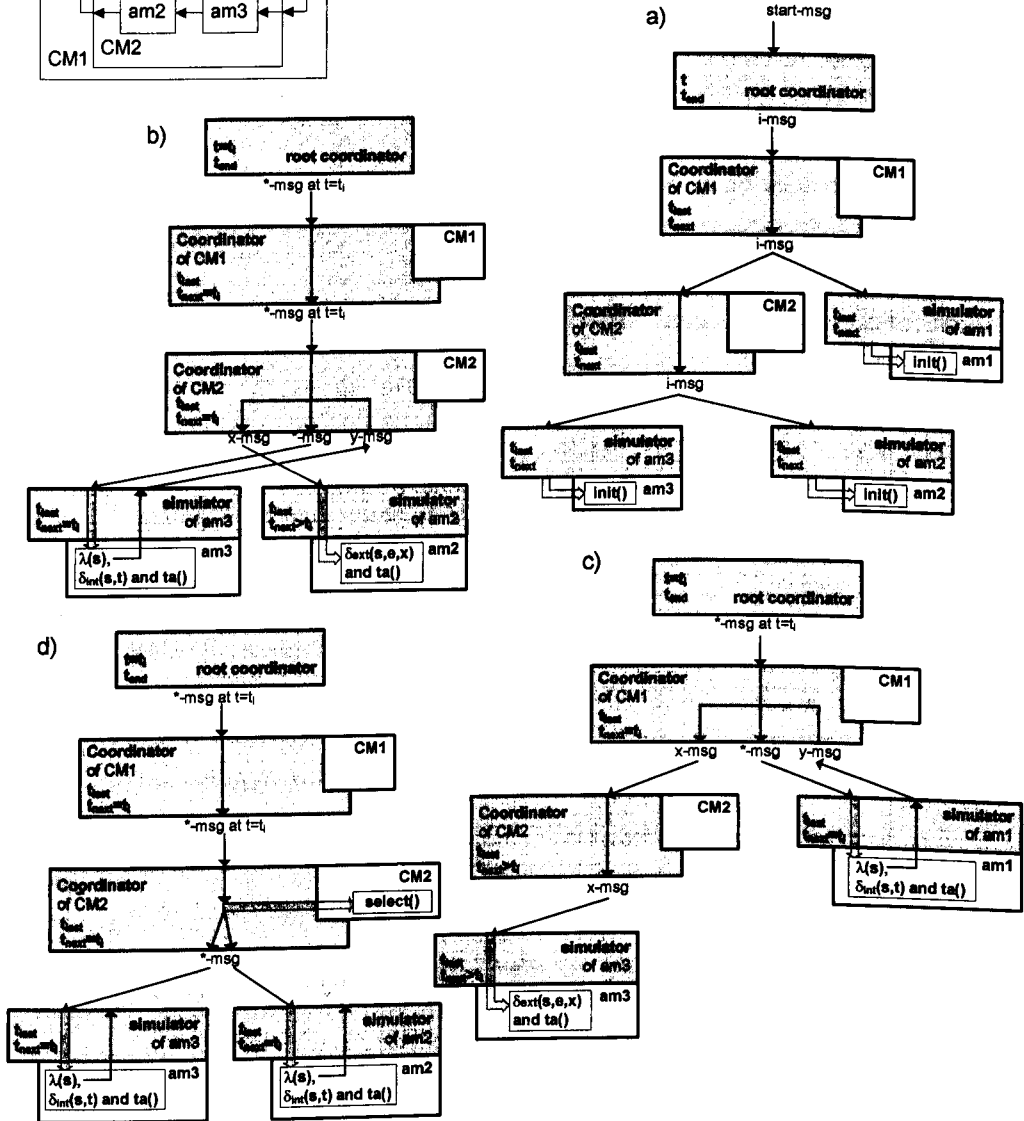
Figure 3.7 An example of a Classic DEVS model with associated abstract simulator elements, messages and model function calls during initialisation and simulation phases

The execution of the simulation model can be subdivided into two phases: initialisation phase and simulation phase. Each phase is started and proceeded by several messages passed between root coordinator, coordinator and simulator instances:

- The initialisation phase starts with an initialisation message (i-msg) generated by the root coordinator. This message is redirected and handled by each coordinator

instance and handled by each simulator instance, respectively. Each simulator instance initialises the internal states S of the associated atomic model and estimates the time of the first next internal event t_{next}. Each coordinator estimates the minimum time of the first next internal events of all sub components. Due to the hierarchical structure of the simulation model the root coordinator instance gets the minimum time of the first internal event of all model components from its direct successor coordinator after a complete i-msg handling.

- The simulation phase is started with the first *- message (*-msg) at the minimum time of next internal event t_{next} estimated by the root coordinator as described above. The consequence of a *-message are subsequent input and output messages (x and y-msg). All simulator instances which received a *- or x-message can change the time of their next internal event t_{next}. All coordinator instances redirecting a *-, x- or y-message estimates the minimum time of next internal events of their sub components. Due to the hierarchical structure of the simulation model the root coordinator instance gets the minimum time of next internal events after a complete *-message handling. The root coordinator instance advances the simulation clock to that time and repeats the complete process by sending the next *-message. Advancing the simulation clock and message handling is repeated in a loop until the simulation end time t_{end} is reached or exceeded.

The different message types created and handled during initialisation and simulation phase have the following characteristics:

- start-msg(tend)
  The start-message is created and sent only once. It starts the simulation model execution with the generation of an i-message.

- i-msg()
  The i-message starts the model component initialisation at time $t=0$. The root coordinator instance sends one i-message to its direct successor coordinator

instance to initialise all model and simulation components. Each coordinator instance sends further i-messages to its sub components.

- *-msg(t)

  A *-message received by a simulator instance starts the processing of an internal event by calling the output function λ, internal state transition function δ_{int} and time advance function ta of the corresponding atomic model. The time of the *-message is stored in t_{last} of the simulator instance. The output of function λ is sent up to the parent coordinator instance as a y-message. The final execution of function ta can cause a new time of the next internal event depending on the internal state S of the atomic model and stored in t_{next} of the simulator instance.

  A *-message received by a coordinator instance is sent to the successor simulator or coordinator instance with the appropriate time t_{next}. For this purpose the coordinator instance compares the actual simulation time with a list of t_{next}-instance pairs. The time-instance-pairs of all next internal events of all sub components are stored in an event chain of the coordinator instance. Concurrent internal events i.e. different sub components have the same t_{next} are resolved by the *select* function of the parent coupled model. After a complete handling of the *-message the coordinator instance estimates the minimum time of next internal events of all sub components and stores it in t_{next}.

- x-msg(t, x)

  An x-message received by a simulator instance calls the external state transition function δ_{ext} and time advance function ta of the corresponding atomic model. The time of the x-message is stored in t_{last} of the simulator instance. The final execution of function ta can cause a new time of next internal event stored in t_{next} of the simulator instance.

  An x-message received by a coordinator instance is redirected to all sub components with an appropriate EIC. After a complete x-message handling the

coordinator instance estimates the minimum time of next internal events of all

sub components and stores it in t_{next}.

- y-msg(t, y)

    The y-message is created by an atomic model/simulator instance. It is routed by

    the super-ordinate coordinator instance according the coupling relations to other

    successor simulator and/or coordinator instances or to the parent of the super-

    ordinate coordinator instance. Receiving simulator or coordinator instances get

    this message as an x-message.

Listings B.3, B.4 and B.5 in appendix B show pseudo codes of Classic DEVS root

coordinator, coordinator and simulator.

3.3   DEVS Extensions

Extensions of the Classic DEVS formalism expand the classes of system models that can be

represented by DEVS. Several DEVS extension are introduced e.g. in [9] [38] [48] [60] [62]

and [66]. At the regular DEVS symposium held at the annual Spring Simulation Multi

Conferences the current development of DEVS, DEVS extensions and DEVS related

developments are published. An incomplete list of DEVS extensions recently presented are:

- DEVS with Ports

    The port extension adds additional input and output ports to atomic and coupled

    models. The approach is introduced later in more detail.

- Parallel DEVS

    Parallel DEVS (PDEVS) considers concurrent transition events. The approach is

    introduced later in more detail.

- Dynamic Structure DEVS

    Dynamic Structure DEVS (DSDEVS) enables model structure changes during a

    simulation run. Several partial very different approaches exist. Dynamic structure

    extensions introduced by Barros [9] and Pawletta et.al. [38] preserve the general

structure of Classic DEVS modelling and simulation with additions to coupled model definitions but unchanged atomic model definitions. Other dynamic structure extensions e.g. Uhrmacher with an agent based DEVS [60] introduce more extensive modifications. The approach of Pawletta et.al. is introduced in more detail in section 3.3.3.

- Symbolic DEVS

  It represents occurring events in a symbolic definition [12]. In conventional DEVS, the time base, its operations and relations are performed with real numbers. In Symbolic DEVS, the objective is to explore multiple model behaviours simultaneously e.g. with a symbolic result of the time advance function [66].

- Real Time DEVS

  The DEVS model is developed in a conventional simulation environment. But it is executed in real time rather than in model time. The time advance function delivers time intervals rather than single values. The interval allows uncertainty when an internal event has to take place.

- Fuzzy DEVS

  Provides another possibility to enable uncertainty into the model set and model function definitions.

The next sections introduce three DEVS extensions in more detail. The chosen extensions are used as a basis of the subsequent unifying DEVS formalism introduced as a key element of this research.

3.3.1 DEVS with Ports

The introduction of ports into the Classic DEVS formalism makes modelling easier and the representation of information flow more clearly [66]. In Classic DEVS each model has only a single input and a single output port. All events are received and sent over these ports. With the port extension, a model has several input and output ports each dedicated for a

specific employment i.e. event type. A model can have several output ports which can be connected to input ports of other models as shown in figure 3.8. Hence, each event can use a dedicated, well defined routing path. The modelling becomes more structured; a model can become clearer and better understandable through differentiated interfaces.
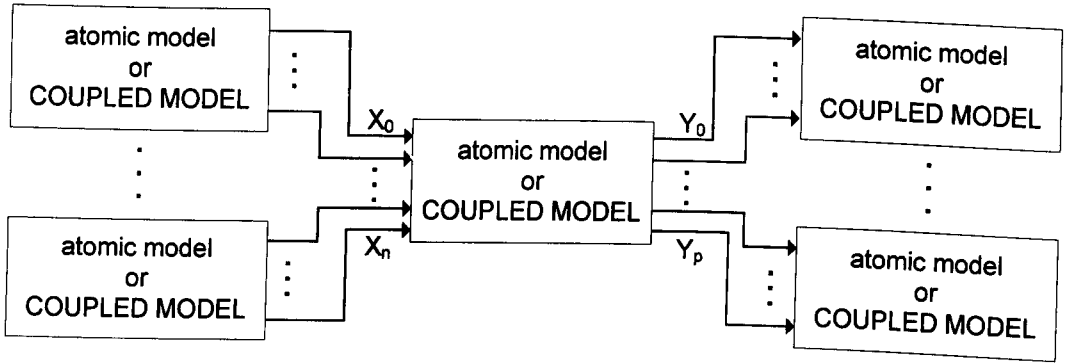
Figure 3.8 Models with multiple input and output ports

The formal description of Classic DEVS with Ports largely remains the same except the extended definitions of X, Y for atomic and coupled models [66]:

$X = \{(p,v) \mid p \in InputPorts, \; v \in X_p\}$

$Y = \{(p,v) \mid p \in OutputPorts, \; v \in Y_p\}$

- p is the input or output port of the model

- v is a discrete value

- X_p and Y_p specify the sets of discrete inputs and outputs at port p

Whereas in Classic DEVS the coupling definitions consist of a sub model name as destination and source, respectively, for EIC and EOC and a pair of sub model names for IC the port extension necessitate a coupling definition extension, too:

- $EIC = \{ (input\_port, d.input\_port) \mid input\_port \in InputPorts, \; d \in D,$

  $d.input\_port \in InputPorts \; of \; M_d \}$

  The external input coupling definition of a coupled model is a set of pairs of an input port name of the coupled model itself and an input port name of the destination sub model.

- $IC = \{ (d_i.output\_port, d_k.input\_port) \mid d_i, d_k \in D, d_i.output\_port \in OutputPorts$ of $M_{d_i}, d_k.input\_port \in InputPorts$ of $M_{d_k}, i <> k \}$

  The internal coupling definition is a set of pairs of an output port name and an input port name of sub models.

- $EOC = \{ (d.output\_port, output\_port) \mid d.output\_port \in OutputPorts$ of $M_d, d \in D, output\_port \in OutputPorts \}$

  The external output coupling definition of a coupled model is a set of pairs of an output port name of source sub component and an output port name of the coupled model itself.

Listings B.6, B.7 and B.8 in appendix B show pseudo codes of an example Classic DEVS with Ports atomic model and pseudo codes of simulator and coordinator. Differences to the Classic DEVS pendants are marked in bold face type.

3.3.2 Parallel DEVS

Parallel DEVS (PDEVS) was introduced by Chow and Zeigler [13]. It adds new elements and functions to the Classic DEVS formalism. It allows all imminent components to be activated and enables sending their output to other components at the same time concurrently. Multiple outputs are combined in a bag which is sent as a whole to a model's external state transition function. A bag is similar to a set, containing an unordered set of elements, but allows multiple occurrences of an element. In Classic DEVS by contrast events are handled individually. In PDEVS during the *-message handling firstly all outputs are established before calling external and internal state transition functions. Each receiving component is responsible for examining and interpreting its combined inputs in the correct order. PDEVS gives the atomic model more control over the handling order of concurrent external and internal events. In Classic DEVS a super-ordinate component, the coupled model, is responsible for the execution order of concurrent internal events of different sub

components using the *select* function. In PDEVS the order of simultaneous events is locally controllable at atomic model level with an additional, third state transition function, the confluent transition function δ_{con}. Hence, it merges the decision logic of execution order of concurrent events with the event handling functions at same level. Apart from that, there is no difference in the principle of event handling to that described in section 3.2.

According to the extensions of PDEVS an atomic model is defined by the following 8- tuple [13]:

$$AM = (X,\ Y,\ S,\ \delta_{ext},\ \delta_{int},\ \delta_{con},\ \lambda,\ ta)$$

- X, Y and S specify the sets of discrete input events, output events and sequential states.

- δ_{ext}: $Q \times X^b \rightarrow S$ where X^b is a bag covering elements of X and $Q = \{\ (s,e)\ |\ s \in S,$

  $0 < e < t_{next}$, *elapsed time* $e = t - t_{last}\ \}$

  The external state transition function δ_{ext} handles a bag covering external inputs $X^b = \{x_i\ |\ x_i \in X\}$.

- δ_{int}: $S \rightarrow S$

  The internal state transition function δ_{int} establishes a new internal state. The execution of output function λ and internal transition function δ_{int} is induced by a time driven internal event. The time of an internal event is established by the time advance function *ta*.

- δ_{con}: $S \times X^b \rightarrow S$

  The confluent transition function δ_{con} handles the execution sequence of δ_{int} and δ_{ext} functions in case of concurrent external and internal events.

  o The definition $\delta_{con}\ (s,\ X^b)\ =\ \delta_{ext}(\delta_{int}(s),\ 0,\ X^b)$ with $\delta_{ext}(s,\ e,\ X^b)$ of the confluent transition function is equivalent to the Classic DEVS behaviour with a higher prioritised internal event handling.

- o The alternative defintion $\delta_{con}(s, X^b) = \delta_{int}(\delta_{ext}(s, ta(s), X^b))$ with $\delta_{int}(s)$ of the confluent function firstly handles external events.

- o The execution of the confluent function with an empty bag $\delta_{con}(s, null)$ calls directly the internal transition function δ_{int}.

- $\lambda: S \rightarrow Y^b$ where Y^b is a bag covering elements of Y

  The output function λ can generate a bag covering outputs $Y^b = \{ y_i \mid y_i \in Y \}$. The generated output depends on the internal state S.

- $ta: S \rightarrow \Re_0^+ \cup \infty$

  The time advance function ta schedules the time of the next internal event after each state transition.

The figure 3.9 shows the dynamic behaviour of an atomic PDEVS model in a situation with concurrent external and internal events. Due to the concurrent events the confluent transition function δ_{con} is called. Depending on the specific implementation of function δ_{con} sequence a) or sequence b) is executed.
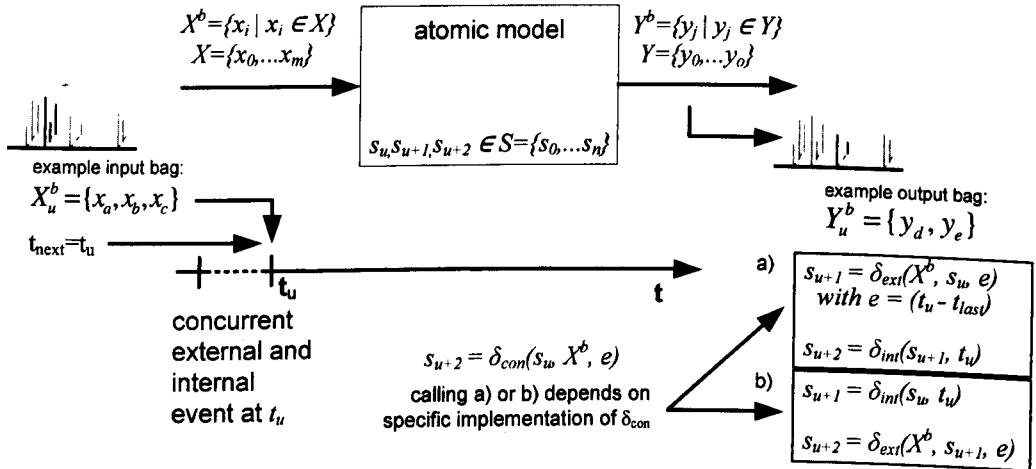
Figure 3.9 Dynamic behaviour of an atomic PDEVS model

The definition of a coupled model for PDEVS is the same as for Classic DEVS except for the absence of the *select* function [13]:

$$CM = (d_n, X, Y, D, \{ M_d \}, EIC, EOC, IC)$$

The generation of an executable PDEVS model is carried out similarly to Classic DEVS i.e. the same coupling of atomic models with simulator instances and coupled models with coordinator instances and the perpetuation of the original hierarchical model structure. Listings B.9 and B.10 in appendix B show pseudo codes of an example PDEVS atomic model and a PDEVS simulator. Differences to the Classic DEVS pendants are marked in bold face type.

3.3.3 Dynamic Structure DEVS

Several approaches extend the Classic DEVS to Dynamic Structure DEVS (DSDEVS). Barros [9] [10] and Pawletta et.al. [42] introduce two DSDEVS variants with an extension of the coupled model definition while the atomic model definition remains unchanged. With theses extensions the coupled model is able to change its structure during simulation time. Uhrmacher et.al. [60] introduce an agent based approach. It defines extensions for both atomic and coupled systems. Another approach is Cell-DEVS, a combination of cellular automata with the DEVS formalism where each cell consists of a single DEVS model [63].

The different types of extensions are carried out due to different application fields or problem definitions e.g. a typical Cell-DEVS application field is social and environmental modelling and simulation. The approaches of Barros and Pawletta are extending the classic formalism without changing its overall principle and thus the general application field of Classic DEVS. This research is restricted to and continues the research of Pawletta. This DSDEVS approach is introduced in detail in the following.

DSDEVS by Pawletta enables several types of structural dynamics:

- creation, destruction, cloning and replacement of sub components

- exchange of a sub component between two coupled models

- changing coupling definitions of a coupled system

Figure 3.10 shows an example of structure changes, the creation of a sub model with an additional extension of the coupling definition.
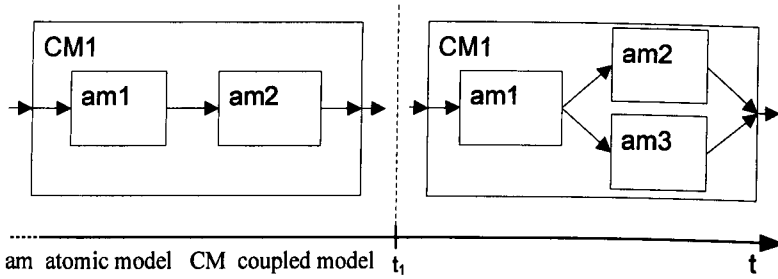
Figure 3.10 Examples of structure changes at coupled model level

Pawletta et.al. have introduced an extension of Classic DEVS to enable structure variability during simulation time [38] ... [45] firstly named Variable Structure DEVS. To avoid name and abbreviation confusions the name of this approach was changed to Dynamic Structure DEVS (DSDEVS) in later publications [34] et seqq. The approach extends the coupled model definition but the atomic model definition stays unchanged. During the simulation time a coupled model can change its structures. Each structure can be seen as a structure state s_i with s_0, s_1, ...,s_n $\in S_{DS}$. A single structure state s_i describes the structure relevant elements of a coupled model i.e. it defines sub components with their couplings, the sets of input and output events together with the concurrent internal event handling function *select*. A structural change of a coupled model means the modification of the current structure state. Additionally a structural state set H_{DS} can store further structure information e.g. the number of structure changes at the present time or the current structure number. External or internal events, handled by the additional state transition functions $\delta_{x\&s}$ and δ_{int} at coupled model level, induce structure state changes and as a result model structure changes. This dynamic structure extension of Classic DEVS was developed with a regard to hybrid systems, i.e. systems with continuous and discrete event dynamics. In the following only the relevant aspects for discrete event systems are taken into account.

A DSDEVS coupled model is defined by the following 6-tuple [38]:

$$CM_{DS} = (d_{ds}, S_{DS}, \delta_{x\&s}, \delta_{int}, \lambda, ta)$$

- d_{ds} specifies the name of the coupled model.

- According to the above definition of a coupled model, its structure consists of sets of sub components and coupling relations. Structure changes means modifications of these sets. Obviously, the sets of sub systems and coupling relations could be interpreted as a structure state. The set of sequential structure states $\{s_0, s_1, ..., s_n\} = S_{DS}$ defines all structure variants of the variable structure coupled model CM_{DS}. Structure state changes can be induced by handling external or internal events of the coupled model itself or by state events i.e. output events of subordinated components. A structure state is defined by a 9-tuple:

$s_i = (X, Y, H_{DS}, D, \{M_d\}, EIC, EOC, IC, select)$

  - X and Y specify the sets of discrete input and output events. The sets exactly match the sets X and Y in Classic DEVS.

  - The set H_{DS} represents additional structure related state variables. They are equivalent to the state set S of an atomic model.

  - D specifies the set of sub component names.

  - $M_d \mid d \in D$

    M_d is the model of the sub component d of the coupled model CM_{DS}. The set $\{M_d\}$ defines all sub components of CM_{DS}.

  - EIC, EOC and IC are the external input, external output and internal couplings.

  - The function *select* prioritises concurrent internal events of the coupled model itself and its sub components.

- $\delta_{x\&s}: Q_{DS} \times X \rightarrow H_{DS}$ where $Q_{DS} = \{(h,e) \mid h \in H_{DS}, 0 < e < t_{next}, \text{ elapsed time } e = t - t_{last}\}$

The external and state transition function $\delta_{x\&s}$ handles external input events and state events i.e. output events of sub components. However it is unreasonable to make changes in the set of sub components or the coupling relations by this function directly. This could lead to ambiguous event handling because external events could

simultaneously influence the dynamic of sub components and the structure state. Consequently the $\delta_{x\&s}$ function is only allowed to modify structure related state variables in the set H_{DS}. However, it can induce a structure state change i.e. a change of the model structure by scheduling an immediate internal event.

- δ_{int}: $S_{DS} \rightarrow S_{DS}$

  The internal transition function δ_{int} can change the structure state s_i to s_{i+1} and as a result induce a structure change of CM_{DS}. The execution of output function λ and internal transition function δ_{int} is induced by a time driven internal event. The time of an internal event is established by the time advance function *ta*.

- λ: $S_{DS} \rightarrow Y$

  The output function λ can generate output events.

- *ta:* $S_{DS} \rightarrow \Re_0^+ \cup \infty$

  As with the dynamic of atomic models, internal events are scheduled by the time advance function *ta*. After each state transition the next internal event is established by the time advance function.

The dynamic behaviour of an atomic model is identical to the behaviour in Classic DEVS. Figure 3.11 shows the dynamic behaviour of a variable structure coupled model. The figure depicts two external input events and one internal event. Reasons for an input event handling can be an external input event at the input port of the coupled model itself or an external output event at the output port of a sub component M_d of the coupled model. The handling of both events by the coupled model is identically. As a result of an event the structure related state variable set H_{DS} can be changed and with the concluding call of the time advance function an immediate internal event can be induced. An internal event is handled by a coupled model similar to the internal event handling of an atomic model, i.e. the event handling can induce a change of the structure state set S_{DS}, and in this case a change in the set of sub components *{M_d}* and/or the coupling sets *EIC, IC* and *EOC*.
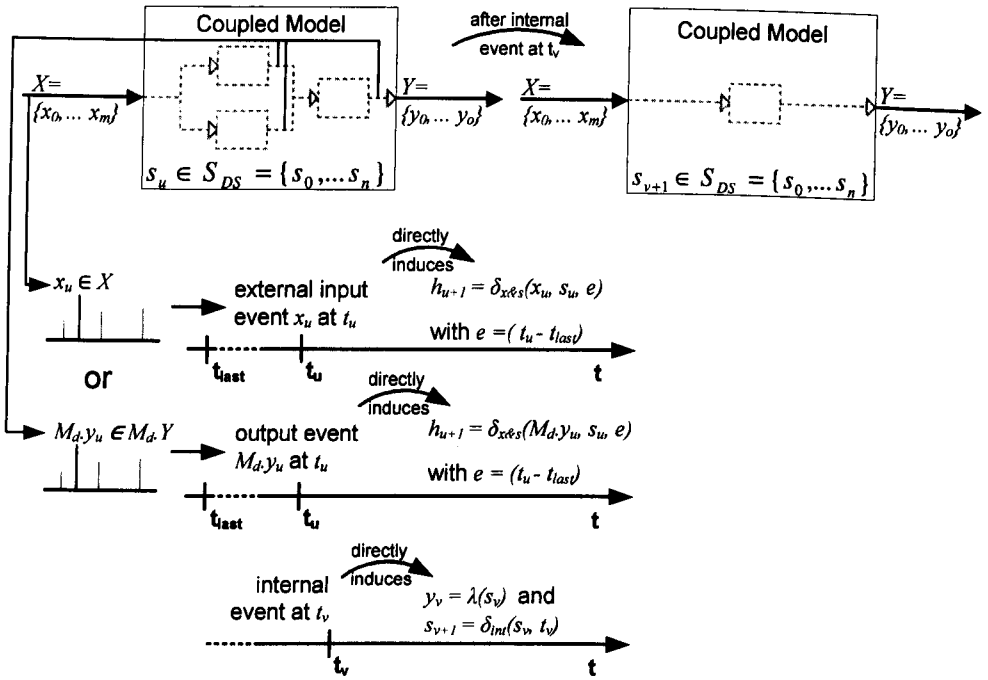
Figure 3.11 Dynamic behaviour of a coupled DSDEVS model

Examples of sequential model structure changes are shown in figure 3.12 a-d. The following definitions of the structure state set describe the insert and change of sub components and couplings as a result of internal events and changes of the sequential structure state set $s_i \in S_{DS}$ by the function δ_{int}. The subsets X, Y and H_{DS} and the *select* function of a structure state $s_i \in S_{DS}$ will not be detailed.
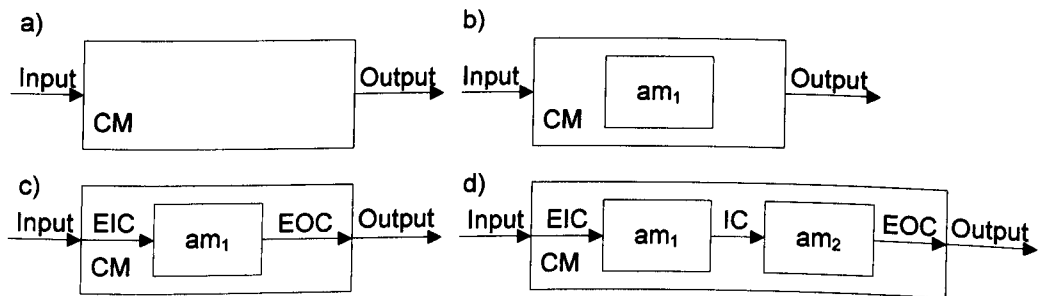
Figure 3.12 Examples of sequential structure changes of a coupled model

a) Figure 3.12a depicts a coupled model *CM* without sub components.

   D, $\{ M_d \}$, *EIC*, *EOC* and *IC* are empty sets

b) In figure 3.12b the coupled model contains one sub component, the atomic model am_1, created as a result of the handling of an internal structure event i.e. the execution of function δ_{int}.

$D = \{ am_1 \}$

$M_d = \{ M_{am_1} \}$

EIC, EOC and *IC* are empty sets

c) Figure 3.12c depicts external input and output couplings created as a result of the handling of an internal structure event i.e. the execution of function δ_{int}.

$D = \{ am_1 \}$

$M_d = \{ M_{am_1} \}$

$EIC = \{ (CM.Input, am_1.Input) \}$

$EOC = \{ (am_1.Output, CM.Output) \}$

IC is an empty set

d) Figure 3.12d depicts the insert of sub component am_2 and the change/creation of several couplings as a result of the handling of an internal structure event i.e. the execution of function δ_{int}.

$D = \{ am_1, am_2 \}$

$M_d = \{ M_{am_1}, M_{am_2} \}$

$EIC = \{ (CM.Input, am_1.Input) \}$

$EOC = \{ (am_2.Output, CM.Output) \}$

$IC = \{ (am_1.Output, am_2.Input) \}$

3.4 Extended Dynamic Structure DEVS

Sections 3.2 and 3.3 introduced the Classic DEVS formalism and several DEVS extensions. Every extension has its advantages and widens the application field of DEVS in a different direction, PDEVS generalises the specification and handling of concurrent events, DEVS with Ports enables a more structured modelling and DSDEVS introduces dynamic structure

changes at coupled model level during simulation time and significantly eases the modelling of larger real systems. The extensions have one joint attribute: they are based on the Classic DEVS formalism. Hence, the decision on one DEVS extension inhibits the use of advantages of another one. This principle leads to the idea of a merging formalism to combine the advantages of different approaches and widen the application field of the resulting formalism. In [66] a first step into this direction is undertaken, the introduced PDEVS formalism is a combination of the original PDEVS and DEVS with Ports. Further steps into this direction are not known. The Extended Dynamic Structure DEVS (EDSDEVS) combines Classic DEVS with the extensions: PDEVS, DSDEVS and DEVS with Ports. The fusion results in a DEVS formalism with the following main characteristics:

- Formal model description by sets and functions

- Exact definition of simulation algorithms

- Modular, hierarchical and dynamic structure modelling and simulation formalism

- Dynamic behaviour description at atomic model level

- Dynamic structure description at coupled model level

- Exact behaviour definition at critical situations with concurrent events

- Substantial similarity between real system and model

The next section introduces the formal concept of EDSDEVS modelling with formal descriptions and dynamic behaviour of atomic and coupled models. Section 3.4.2 goes into detail of the EDSDEVS simulation concept with abstract simulator algorithms, message handling and model function calls.

3.4.1 Formal Concept of EDSDEVS Modelling

The EDSDEVS formal descriptions of coupled and atomic models as a combination of sets and functions are similar structured as the Classic DEVS formal description as introduced in section 3.2.2.

An atomic EDSDEVS model is a fusion of PDEVS with DEVS with Port atomic model definitions. The atomic EDSDEVS model AM_{EDS} is defined as an 8- tuple:

$$AM_{EDS} = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

- $X = \{(p,v) \mid p \in InputPorts, v \in X_p\}$

  $Y = \{(p,v) \mid p \in OutputPorts, v \in Y_p\}$

  The definitions of both sets are identical to the definitions in DEVS with Ports as introduced in section 3.3.1.

- S specifies the set of internal states and is identical to internal state set S of an atomic Classic DEVS model.

- δ_{ext}: $Q \times X^b \rightarrow S$ with $X^b = \{x_i \mid x_i = (p,v), p \in InputPorts, v \in X_p\}$ and

  $Q = \{(s,e) \mid s \in S, 0 < e < t_{next}, elapsed\ time\ e = t - t_{last}\}$

  The external state transition function δ_{ext} handles a bag covering external inputs. Each input consists of a pair of a discrete input $v \in X_p$ and an input port $p \in InputPorts$. The set X_P is the set of discrete inputs at port p and $InputPorts$ is the set of input ports of model AM. The function δ_{ext} can induce an internal event with a rescheduling of the time of the next internal event.

  This extended definition of δ_{ext} is a fusion of the δ_{ext} definitions of PDEVS and DEVS with Port.

- δ_{int}: $S \rightarrow S$

  The internal state transition function δ_{int} can establish a new internal state. The execution of output function λ and internal state transition function δ_{int} is induced by a time driven internal event. The time of an internal event is established by the time advance function ta.

  The definition is identical to definition in Classic DEVS.

- δ_{con}: $S \times X^b \rightarrow S$

The confluent transition function δ_{con} handles the execution order of δ_{int} and δ_{ext} functions during concurrent external and internal events. In spite of the same function signature $\delta_{con}(s, X^b)$ the parameter X^b is different to that in the PDEVS definition as described in section 3.3.2. Anyhow the three δ_{con} definitions also apply here.

This extended definition of δ_{con} is based on the PDEVS δ_{con} function definition. Unlike in PDEVS the function has to handle a bag covering inputs. Each input consists of a pair of discrete input and input port.

- $\lambda: S \rightarrow Y^b$ with $Y^b = \{y_i \mid y_i = (p, v),\ p \in OutputPorts, v \in Y_p\}$

The output function λ can generate a bag covering outputs Y^b. In spite of the same function signature $Y^b = \lambda\ (s)$ the function result Y^b is different to that in the PDEVS definition as described in section 3.3.2. The function result is a bag covering outputs $Y^b = \{\ y_i \mid y_i = (p, v)\ \}$ each consisting of a pair of discrete output $v \in Y_p$ and output port $p \in OutputPorts$. The set Y_P is the set of discrete outputs at port p and $OutputPorts$ is the set of output ports of model AM. If and which outputs are generated depends on the internal state S.

This extended definition of λ is based on the PDEVS λ function definition. Unlike in PDEVS the function generates a bag covering outputs each consisting of pairs of discrete output and output port as introduced in DEVS with Ports.

- $ta: S \rightarrow \Re_0^+ \cup \infty$

The time advance function ta schedules the time of the next internal event after each state transition. The definition is identical to the definition in Classic DEVS as introduced in section 3.2.2.

The figure 3.13 shows the dynamic behaviour of an atomic EDSDEVS model am_{EDS}. At time t_u the confluent transition function δ_{con} handles two concurrent events. The first event contains a bag covering external inputs received by the atomic model am_{EDS}. The figure

depicts an example bag covering three external inputs received at two different input ports. A concurrent internal event at t_u was scheduled by the previous execution of the time advance function ta. Depending on the specific implementation of function δ_{con} sequence a) or sequence b) is executed. The execution of the output function λ creates a bag covering outputs. The depicted example bag Y_u^b covers two outputs at two different output ports.
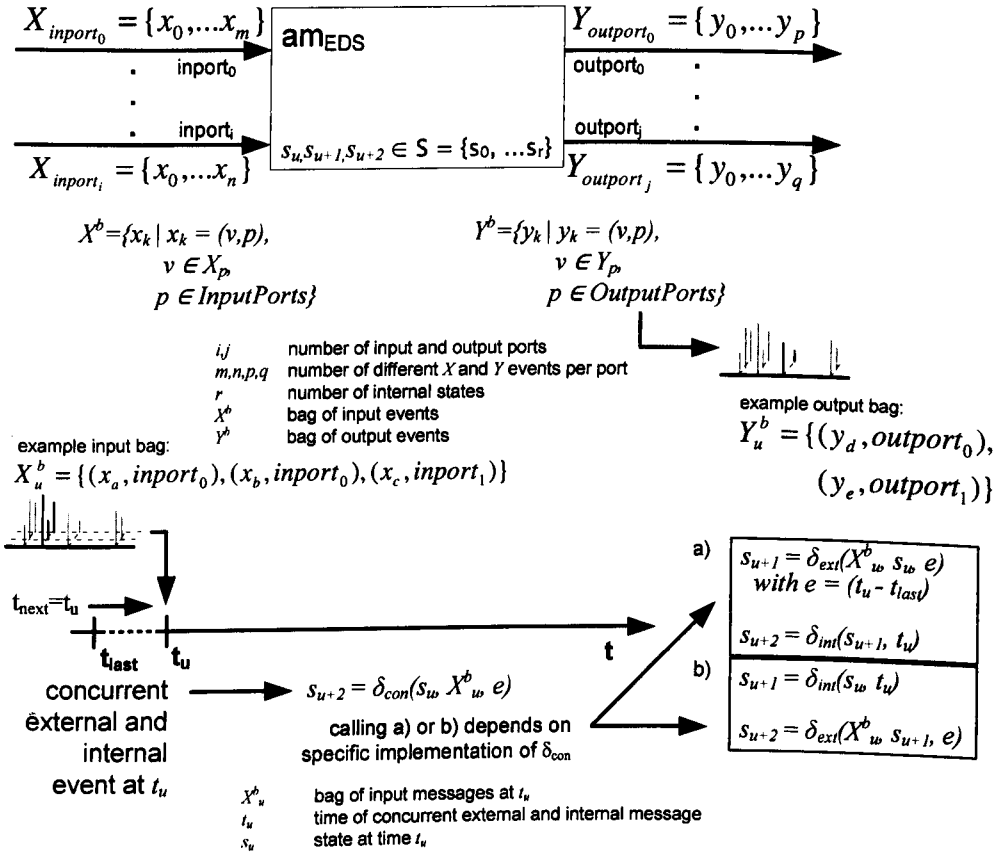
Figure 3.13 Dynamic behaviour of an atomic EDSDEVS model

Listings B.11 in appendix B shows pseudo code of an atomic EDSDEVS model.

A coupled EDSDEVS model is defined by the following 7-tuple:

$$CM_{EDS} = (d_{EDS}, \ S_{EDS}, \ \delta_{x\&s}, \ \delta_{int}, \ \delta_{con}, \ \lambda, \ ta)$$

- d_{EDS} specifies the name of the coupled model.

- In the EDSDEVS formalism the coupled model structure consists not only of sets of sub components and coupling relations as in DSDEVS, introduced in section 3.3.3,

[60]

but also of additional interface definitions i.e. input and output port definitions. The set of sequential structure states $\{s_0, s_1, ...,s_n\} = S_{EDS}$ has to define all structure variants of the coupled model CM_{EDS}. Two model structure variants can vary in different interface definitions, in contrast to DSDEVS where each model has a non-variable interface with a single input and a single output port. Hence, a structure state has to incorporate interface definitions with sets of input and output ports additionally to the structure state definition as introduced in section 3.3.3. An EDSDEVS structure state is defined by a 10-tuple:

$s_i = (X, Y, H_{EDS}, D, \{ M_d \}, InputPorts, OutputPorts, EIC, EOC, IC)$

- X and Y specify the sets of discrete input and outputs. The sets exactly match the extended definitions of X and Y as introduced in section 3.3.1 with the introduction of DEVS with Ports.

- The sets H_{EDS}, D and M_d exactly match the sets H_{DS}, D and M_d of the DSDEVS formalism introduced in section 3.3.3.

- $InputPorts$ and $OutputPorts$ specify the sets of input and output port names of the coupled model CM_{EDS}. These two elements of the structure state s_i are introduced by the EDSDEVS formalism.

- EIC, EOC and IC are the external input, external output and internal couplings of CM_{EDS}. The definition of the coupling relations exactly match the definition as introduced with the DEVS with Ports extension in section 3.3.1.

- $\delta_{x\&s}$: $Q \times X^b \to H_{EDS}$ where X^b is a bag covering input, input port pairs and

$Q = \{(h,e) \mid h \in H_{EDS}, 0<e<t_{next}, \text{ elapsed time } e = t - t_{last} \}$

The external and state transition function δ_{ext} handles a bag covering inputs. Each input consists of a pair of:

○ a discrete input $v \in X_p$ and an input port $p \in InputPorts$. The set X_P is the set of discrete inputs at port p and $InputPorts$ is the set of input ports of model CM_{EDS}.

○ a discrete output $v \in M_d.Y_p$ and an output port $p \in M_d.OutputPorts$ where M_d is the model of the sub component d of the coupled model CM_{EDS}. The set $M_d.Y_P$ is the set of discrete outputs at port p and $M_d.OutputPorts$ is the set of output ports of model M_d.

○ a discrete input $v \in M_d.X_p$ and an input port $p \in M_d.InputPorts$ where M_d is the model of the sub component d of the coupled model CM_{EDS}. The set $M_d.X_P$ is the set of discrete inputs at port p and $M_d.InputPorts$ is the set of input ports of model M_d.

This extended definition of δ_{ext} is a fusion and extension of the δ_{ext} definitions of DSDEVS, PDEVS and DEVS with Ports. In DSDEVS only state events induced by output events of sub components are handled. However, an output port can have coupling relations to multiple input ports. In this case there is a difference in the handling of a single output event of a single source sub model or multiple input events of different destination sub models. Hence, the external and state transition function of EDSDEVS can handle both output and input events. However, the functionality is in accordance with the description of the DSDEVS external and state transition function $\delta_{x\&s}$ introduced in section 3.3.3.

• $\delta_{int}: S_{EDS} \rightarrow S_{EDS}$

$ta: S_N \rightarrow \Re_0^+ \cup \infty$

The internal state transition function δ_{int} and the time advance function ta exactly match the functions of the DSDEVS formalism introduced in section 3.3.3.

• $\delta_{con}: S_{EDS} \times X^b \rightarrow S_{EDS}$

The confluent transition function δ_{con} handles the execution sequence of δ_{int} and δ_{ext} functions during concurrent external and internal events.

The EDSDEVS formalism introduces the confluent transition function also at coupled model level due to the fusion of PDEVS and DSDEVS. A coupled EDSDEVS model handles external, state and internal events itself instead of only forwarding them as in PDEVS. Hence and in contrast to PDEVS, in EDSDEVS concurrent external and internal events can occur also at coupled model level. Consequently, a confluent transition function to handle concurrent events is also necessary at this level. The functionality is in accordance with the description of the confluent transition function δ_{con} for atomic model in this section.

- $\lambda: S_{EDS} \rightarrow Y^b$

  The output function λ can generate a bag covering outputs $Y^b = \{y_i\}$. An output y_i consists of a pair of discrete output $v \in Y_p$ and output port $p \in OutputPorts$. The set Y_P is the set of discrete outputs at port p and $OutputPorts$ is the set of output ports of model CM_{EDS}. If and which output event is generated depends on the internal state S_{EDS}.

  The output function λ in the EDSDEVS formalism merges three sources:

  o The output function λ at coupled model level is introduced by DSDEVS.

  o The definition of the function creating a bag covering outputs is based on PDEVS.

  o The output event structure with pairs of output/output port is introduced by DEVS with Ports.

The figure 3.14 shows the dynamic behaviour of a coupled EDSDEVS model CM_{EDS}. At time t_u the confluent transition function δ_{con} handles concurrent external and internal events. The first event is a bag covering inputs received at input ports by the coupled model CM_{EDS}. The figure depicts an example bag covering three external inputs received at two different input ports. A concurrent internal event at t_u was scheduled by the last execution of the time

advance function. Depending on the specific implementation of function δ_{con} sequence a) or

sequence b) is executed. The execution of the internal state transition function δ_{int} can change

the structure state s_u to s_{u+1} or s_{u+1} to s_{u+2} and therefore the model structure of CM_{EDS} to

CM^*_{EDS}. The execution of the output function λ creates a bag covering outputs Y_u^b. The

depicted example bag Y_u^b covers two outputs at two different output ports.

Listings B.12 in appendix B shows pseudo code of a coupled EDSDEVS model.
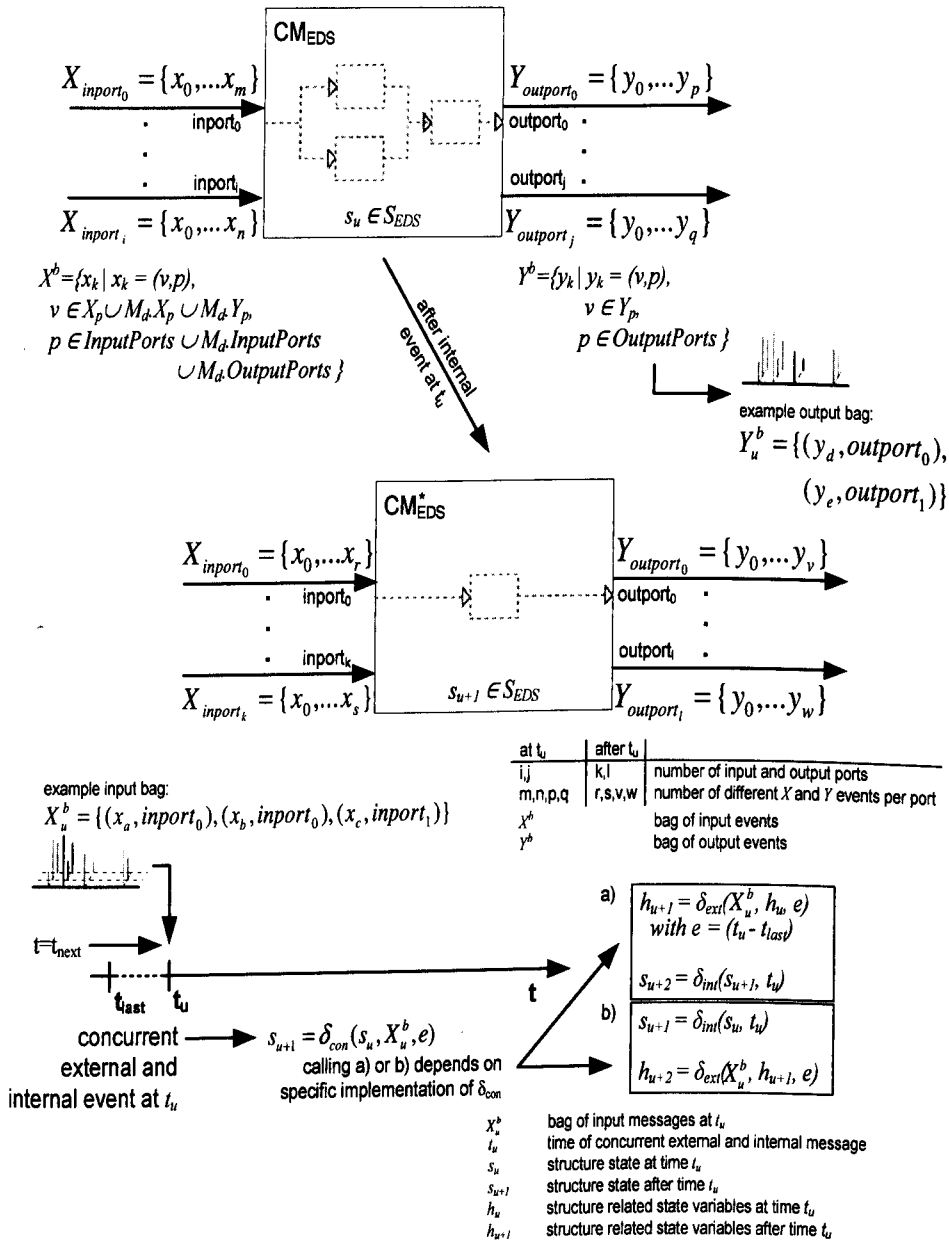
Figure 3.14 Dynamic behaviour of a coupled EDSDEVS model

3.4.2 EDSDEV Simulation

The simulation engine for EDSDEVS models is a combination and extension of the simulation algorithms of Classic DEVS, PDEVS and DSDEVS. The message handling of coordinators are largely similar to simulators. Each coordinator holds its own time of next internal event in t_{next_c} and searches the minimum time of next internal event in t_{next} of sub components and in its own t_{next_c}.

Figures 3.15 and 16 depict an EDSDEVS model example with the associated simulation model components i.e. root coordinator, coordinator and simulator instances and the message handling. The figure is based on and extends figure 3.7 depicting a Classic DEVS model example with associated simulation model components and message handling. The overall structure is very similar to the Classic DEVS simulation model execution except for additions at the levels of coordinator and associated coupled model. Because of complexity and clarity selected situations are shown in sections:

i.   (Figure 3.15a) initialisation phase with i-message handling:

During the initialisation phase model component's init functions are called because of an i-message handling similar to Classic DEVS. Additionally, after structure changes i.e. modification of the sub component set during the simulation phase the init function is called too.

ii. (Figure 3.16b) *-message handling created due to an internal event of model am2:

The root coordinator advances the simulation clock and a *-message is firstly created. The message is sent to the successor coordinator instance of coupled model *CM1* (not depicted). This coordinator instance compares the actual simulation time t with its own next internal event time stored in t_{next_c} and determines that it is not responsible for handling this event. Hence, the event is forwarded to the successor coordinator instance of *CM2*. The coordinator instance is again not responsible for handling the message itself but knows that a sub component scheduled the event. The coordinator instance will then forward the message to the appropriate simulator

instance associated with *am2*. The simulator instance of *am2* calls the model functions λ and δ_{int}. A result of calling λ could be a y-message sent back to the subordinate coodinator instance of *CM2*. This coordinator instance reacts with the call of the model function $\delta_{x\&s}$ of *CM2* and a messge forward to the simulator instance of *am3* due to an appropriate IC coupling.

iii. (Figure 3.16c) *-message handling created due to an internal event of model CM2:

The depicted situation is similar to 3.16b except that the coordinator instance of *CM2* determines that simulation time t and its t_{next_c} are equal. Hence, it has to handle the *-message itself with calling λ and δ_{int} model functions of *CM2* with the possibility of generating a y-message sent to a sub component and/or superordinated coordinator instance and of changing its sequential structure state S_{EDS}.

iv. (Figure 3.16d) concurrent event handling with the confluent transition function δ_{con}:

The figure depicts the handling of concurrent external and internal messages by the coodinator instance of *CM2*. The confluent function of *CM2* is called to handle the concurrent messages. Depending on the specific implementation of δ_{con} the external transition function $\delta_{x\&s}$ and internal transition/output functions δ_{int}, respectively, are firstly called.The external message is concurrently handled by the function δ_{con} and forwarded to the simulator instance of sub component *am2* as a x-message due to an appropriate EIC. Calling the output function λ could cause a y-message sent to a sub component and/or superordinated coordinator instance.

v. (Figure 3.16e) x-message handling:

(i) x-message at $input_0$ of CM2 and due to an appropriate EIC at $input_0$ of am2:

The first x-message is received by the coordinator instance of *CM2*. This message is handled by the function $\delta_{x\&s}$ of the coupled model itself and concurrently forwarded to the simulator instance *am2* due to an appropriate EIC. Because no concurrent internal event exists the function δ_{con} is not called.

(ii) y-message at $output_0$ of am2 and due to an appropriate IC forwarded as x-message to $input_0$ of am3:

Due to an internal event the model *am2* generates a y-message. This y-message is handled by the super-ordinate coordinator instance which calls the function $\delta_{x\&s}$ of its associated model *CM2*. The coordinator instance concurrently forwards the y-message as an x-message to the simulator instance of *am3* because an IC exists between the output port *$output_0$* of *am2* and the input port *$input_0$* of *am3*.
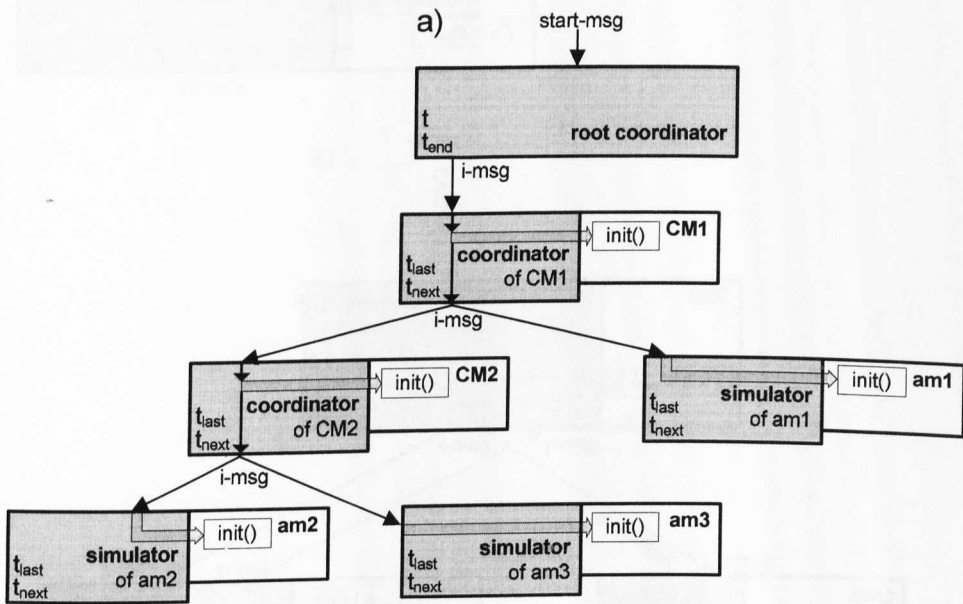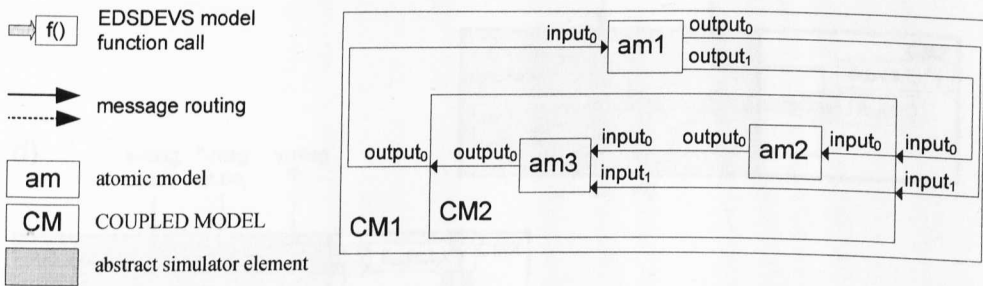
Figure 3.15 An EDSDEVS model example with associated abstract simulator elements, messages and model function calls during initialisation phase
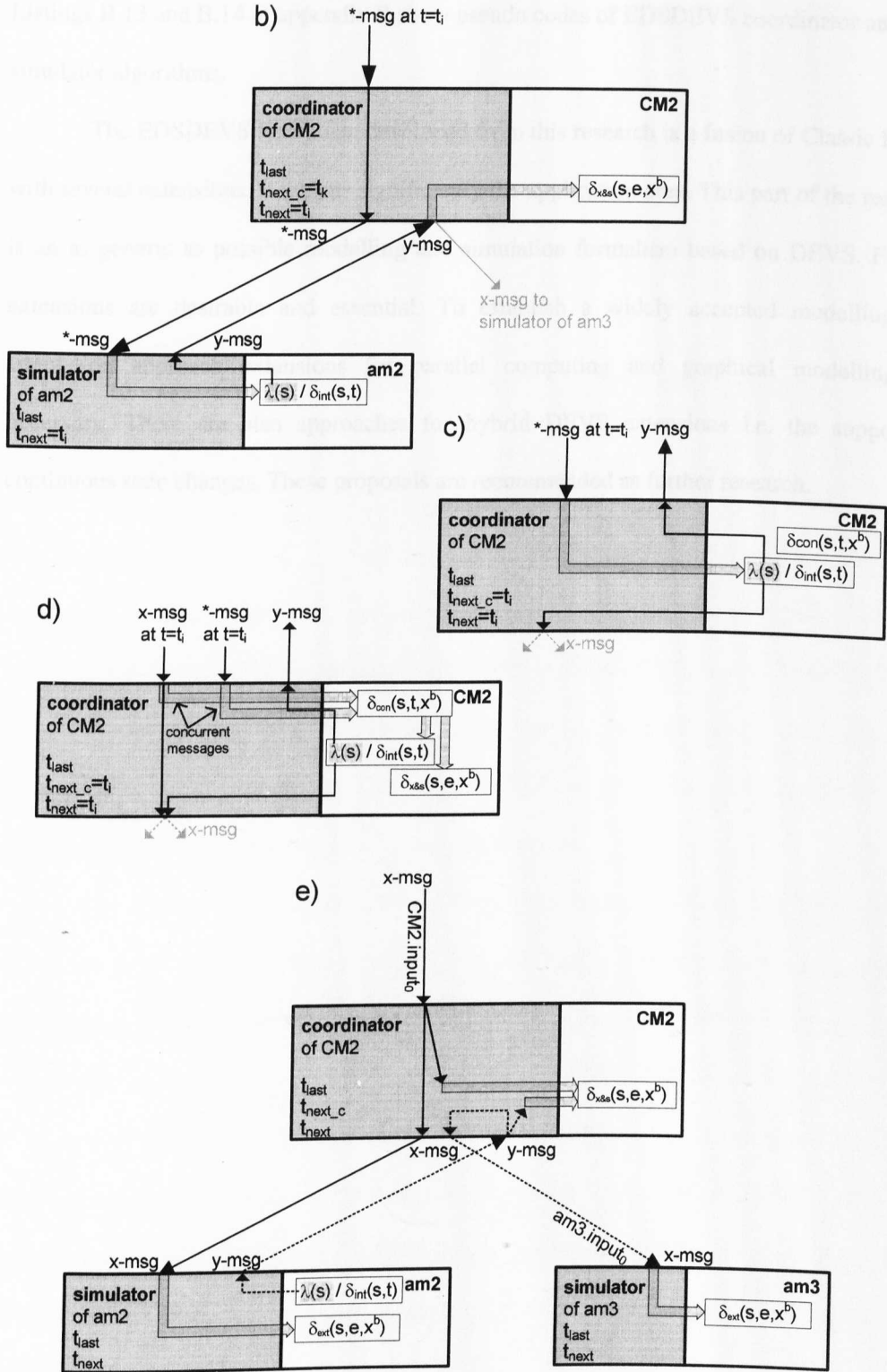
Figure 3.16 An EDSDEVS model example with associated abstract simulator elements,

messages and model function calls during simulation phase

Listings B.13 and B.14 in appendix B show pseudo codes of EDSDEVS coordinator and simulator algorithms.

The EDSDEVS formalism developed from this research is a fusion of Classic DEVS with several extensions. It widens significantly the application area. This part of the research is an as generic as possible modelling and simulation formalism based on DEVS. Further extensions are desirable and essential. To establish a widely accepted modelling and simulation approach extensions for parallel computing and graphical modelling are necessary. There are also approaches for hybrid DEVS extensions i.e. the support of continuous state changes. These proposals are recommended as further research.

Chapter 4

Model Management – Model Set Specification and Organisation

Zeigler introduced in [66] a simulation based system design approach. It is a plan – generation – evaluation process. The *plan* phase organises design alternatives with different model structures and model parameters within defined system boundaries to satisfy given design objectives. During the *generation* phase a specific model design is chosen and the corresponding model is generated. This model is simulated during the *evaluation* phase using an experimental frame derived from the design objectives.

The System Entity Structure/Model Base framework (SES/MB) [52] [66] is such a simulation based system design approach. It is specifically configured to define, organise and generate modular, hierarchical models and was developed to assist an analyst in model organisation and generation. To represent a set of modular, hierarchical models, the SES/MB framework is able to describe three relationships: decomposition, taxonomy and coupling. *Decomposition* means the formalism is able to decompose a system object called 'entity' into sub-entities. *Taxonomy* means the ability to represent several possible variants of an entity called 'specialisation'. To interconnect sub-entities the definition of a *coupling* relationship is necessary.

The literature e.g. [52], [65] [66] and [69] describes slightly different specifications of the SES/MB framework. Hence, section 4.1 defines a classic SES/MB framework according to [52] and [66] as a basis for further extensions introduced in section 4.2.

4.1 Classic System Entity Structure/Model Base Framework

The SES/MB framework approach is [52] [66]:

- The framework consists of two parts: (i) the system entity structure and (ii) the model base.

- A modular, hierarchical model is constructed based on: (i) the declarative system knowledge coded in a SES and (ii) predefined basic system models stored in a MB.

- The partitioning of a modular, hierarchical model is highly dependent on the design objectives. Model parameters are a typical example. They are not really a part of the model composition structure but nevertheless they can become a part of the system entity structure if they are crucial for describing design alternatives.

- The model generation from a SES/MB is a multistage process. The first step is a graph analysing and pruning process to extract a specific system configuration. Based on this information a modular, hierarchical model is generated.

The SES is represented by a tree structure containing alternative edges starting at decision nodes. With the aid of different edge types and decision nodes a set of different model variants can be defined. To choose a specific design and to create a specific model variant the SES has to be pruned. The pruning process decides at decision nodes which alternative(s) to chose as a consequence of specified structure conditions and selection rules. The result of this process is a Pruned Entity Structure (PES) that defines one model variant. A composition tree is derived from a PES. The composition tree contains all the necessary information to generate a modular hierarchical model using predefined basic components from the model base (MB). Figure 4.1 shows the principal organisation and the transformation process: SES → PES → Composition Tree + MB → Modular, Hierarchical Model.
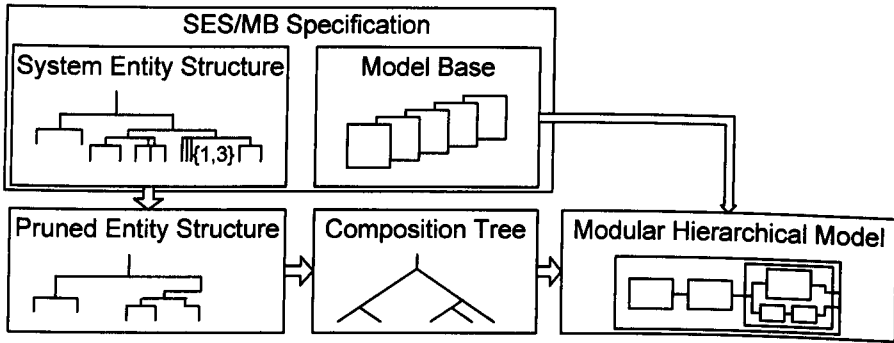
Figure 4.1 SES/MB formalism based model generation

The used SES definition is based on definitions published in [52] and [66]. A SES is a labelled tree consisting of different nodes with optional properties and different edge types. Figure 4.2 depicts a SES example which is referenced by the definition.
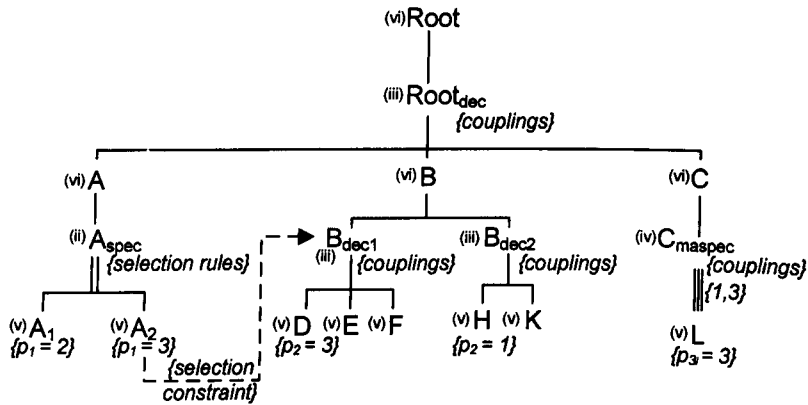
Figure 4.2 A SES example

The SES formalism differentiates four types of nodes: *(i) entity, (ii) specialisation, (iii) aspect* and *(iv) multi-aspect.* An entity node represents a system object. There are two subtypes of entity nodes – *(v) atomic entity* and *(vi) composite entity.* An *atomic entity* (figure 4.2 *(v))* cannot be broken down into sub-entities. The model base contains a corresponding model for each *atomic entity.* Atomic models (described in chapter 3) and *atomic entities* must not be mixed at this point i.e. an *atomic entity* can also correspond to a coupled model in the model base. A *composite entity* (figure 4.2 *(vi))* is defined in terms of other entities, which can be of type *atomic* or *composite entity.* Thus, the root node of a tree is always of type *composite entity*, while all leaf nodes are always of type *atomic entity.* The

root node and each *composite entity* node of the tree has at least one successor node of type -

specialisation (figure 4.2 *(ii)*), *aspect* (figure 4.2 *(iii)*) or *multiple-aspect* (figure 4.2 *(iv)*).

That means there is an alternate mode between entity nodes and the other node types. The

definition of the different node types can be briefly summarised as follows:

atomic entity node = (name, {av$_1$,... av$_n$},selection constraints}

composite entity node = (name, successors, {av$_1$,... av$_n$})

An entity node is defined by a name and is of type *atomic* or *composite*. Both node types

may have attached variables *av*. A *composite entity node* can have a single successor node of

type *specialisation* or *multi-aspect* or multiple successor nodes of type *aspect*. An *atomic*

entity node can have attached *selection constraints* when it is a successor of a *specialisation*

node.

specialisation node = (name, successors, selection rules)

A s*pecialisation node* is defined by a name and a set of successor nodes. In the tree it is

indicated by a double-line edge. A *specialisation node* defines the taxonomy of a

predecessor entity node and specifies how the entity can be categorised into specialised

entities. A *specialisation node* always has successor nodes of type atomic entity to represent

the possible specialisations. A *specialisation node* can define additional *selection rules* to

control the way in which a specialised entity is selected during the pruning process. *Selection*

constraints are added to successor entity nodes of a s*pecialisation node*. The s*pecialisation*

node A in figure 4.2 has two specialisations defined by the nodes A_1 and A_2. During the

pruning process one of these specialisations is chosen. Due to the s*election rule* at node A_2 it

is mandatory to chose node B_{decl} when node A_2 is chosen.

aspect node = (name, successors, coupling specification)

An *aspect node* is defined by a name, a set of successor nodes and coupling information. It is

indicated by a single-line edge in a SES tree. An *aspect node* defines a single possible

decomposition of its parent node and can have multiple successors of type atomic and/or composite entity. The *coupling specification* is a set of couplings and describes how the sub-entities, represented by the successor nodes, have to be connected. Each coupling is defined by a 2-tuple. Each tuple consists of sub-entity source and destination information, e.g. (SourceEntity.outputport, DestinationEntity.inputport). The composite entity B in figure 4.2 has two decomposition variants defined by the *aspect nodes* B_{dec1} and B_{dec2}. During the pruning process one of the decomposition variants has to be chosen.

Using SES/MB to describe a DEVS model an *aspect node* defines the composition of a coupled model.

multiple aspect node = (name, successor, coupling specification, number range property)

The definition of a *multiple aspect node* is similar to an aspect node. However, it defines additionally a *number range property* and has only one successor node of type atomic entity. It is indicated by a triple-line edge in a SES tree. A *multiple aspect node* also defines a decomposition of a composite entity, but all sub-entities have to be of the same entity. Only the number of sub-entities is variable according to the attached *number range property*. The *multiple aspect node* C_{maspec} in figure 4.2 illustrates the decomposition of composite entity C that may be composed by one, two or three sub-entities L.

A *multiple aspect node* also defines the composition of a coupled model.

In figure 4.3 a SES/MB example points up the complete process of model generation from a SES/MB to a modular hierarchical model. The SES tree defines a coupled model *CM1* with two structure variants. The two variants are defined by the specialisation node *CM2_spec* and specialisations *CM2.1* and *CM2.2*. The model base contains several basic components which are referenced by the SES. The different possible pruning results are PES *variant1* and *variant2*. After a transformation to a composition tree and a model generation, with the basic components taken from the model base, the final results are the modular hierarchical model

variant1 and *variant2*, respectively. The SES tree does not define selection rules or selection constraints. Hence, an analyst has to use other, external criteria to decide which alternative structure should be chosen during the pruning process.
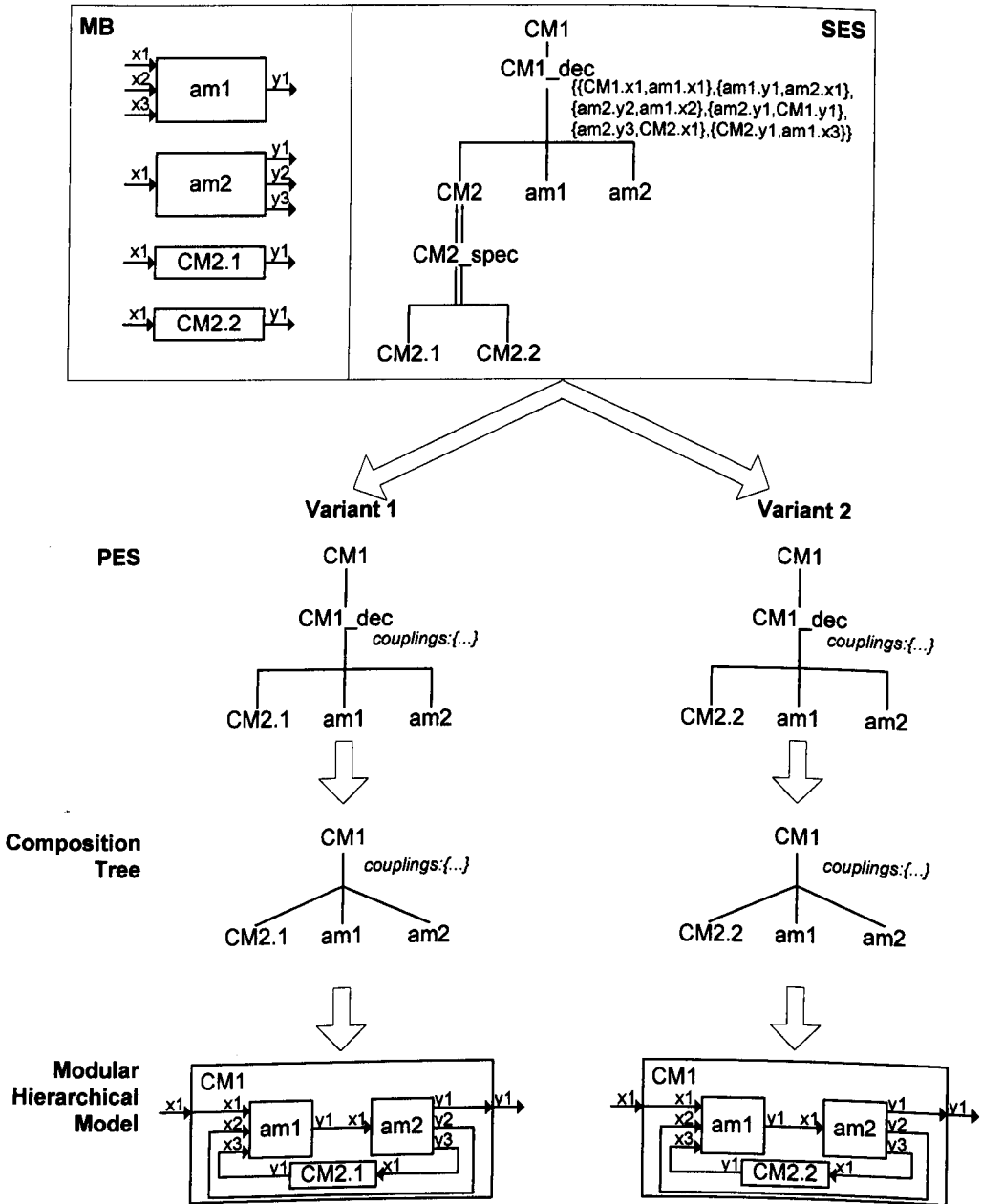
Figure 4.3 Detailed pruning and model generation example

4.2 Extension of the System Entity Structure/Model Base Framework

Originally the SES/MB framework was developed to assist an analyst during the model variant selection and a subsequent model generation. Pruning as a part of these processes is a stepwise procedure with decisions at decision nodes under the control of selection rules and structure constraints. Both rules and constraints represent supplementary structure-knowledge as an addition to the structure-knowledge coded in the SES tree. The supplementary structure-knowledge is used to support the selection of design alternatives and to avoid invalid structures. This knowledge representation is customised to its usage during the pruning. The upper part of figure 4.4 depicts the steps of the original pruning process. An analyst initialises attached variables and makes decisions as long as unpruned decision nodes exist. A decision at a specific decision node can cause the pruning at other nodes according selection rules and structure constraints. The pruning in classic SES is a n-step procedure (n is equal or less than the number of decision nodes) with the goal to synthesise one valid model configuration.

In this research a new pruning principle is introduced. The lower part of figure 4.4 depicts the steps of the new pruning process. The new process is based on information delivered by the optimisation method as depicted in figure 2.5 and is carried out in a single step. A structure validation based on structure-knowledge is carried out after the pruning – not during – as in the original SES/MB framework. This important development means that the new pruning procedure requires another representation for structure-knowledge originally coded in selection rules and structure constraints. The new pruning of a SES tree is carried out in one step based on the structure parameter set X_{Si}. The model structure is verified in a second, following step. The new pruning algorithm is a 2-step procedure. Figure 4.4 identifies the differences between the original and new principle. A detailed description of the new approach is given in chapter 5.

Structure conditions as a new, alternative structure-knowledge representation are added to composite entity nodes. They are used as the alternative to selection rules and

structure constraints as defined in [52] and [66]. During the pruning sub trees are removed. The remaining *structure conditions* are evaluated to verify the PES. Only if all *structure conditions* are true the PES is valid.

Original Pruning

```
                        Start
                          │
                          ▼
  ┌───────────────────────────────────────────────────────────────────┐
  │ Pruning   ┌──►  Repeat until no unpruned decision nodes exist.  ┐  │
  │           │                                                      │  │
  │        ◇ Exists                              ┌──────────────────┐│  │
  │        an unpruned   Yes   ┌──────────────┐  │ Check selection  ││  │
  │        decision     ────►  │ Make a       │─►│ rules and        ││  │
  │        node?               │ decision     │  │ structure        ││  │
  │                            │ at a decision│  │ constraints. If  ││  │
  │                            │ node.        │  │ necessary        ││  │
  │           │ No             └──────────────┘  │ automatically    ││  │
  │           │                                  │ prune appropriate││  │
  │           ▼                                  │ decision nodes.  ││  │
  └───────────────────────────────────────────────────────────────────┘
                          │
                          ▼
                         PES
```

New Pruning

```
                     Start
                       │
                       ▼
      ┌──────►  ┌──────────────────┐
      │         │  Establish X_Si  │
      │         └──────────────────┘
      │                 │
      │                 ▼
      │         ┌──────────────────┐
      │         │     Pruning      │
      │         └──────────────────┘
      │                 │
      │                 ▼
      │            ◇ Check
    Invalid        Structure
      └──────────  Conditions
                       │
                  Valid│
                       ▼
                      PES
```
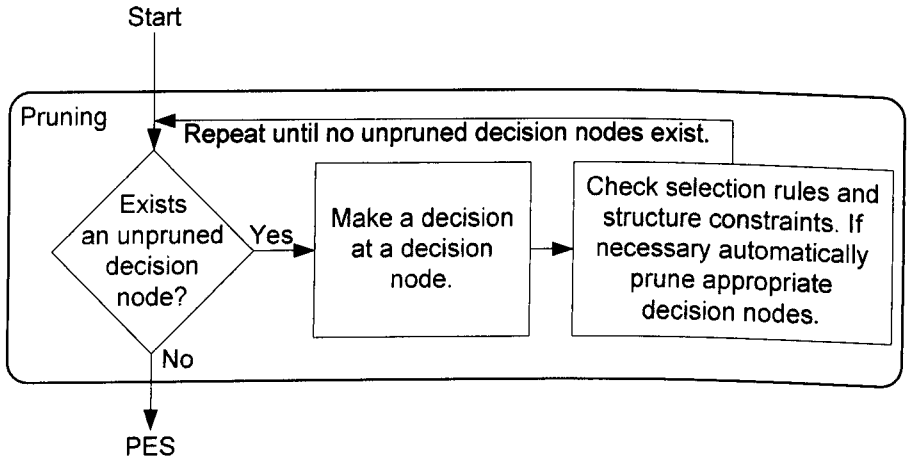
Figure 4.4 Comparison original pruning – new pruning principle

Figure 4.5 shows an example SES with a *structure condition* added to the composite entity node *ROOT*. The SES defines 12 different design variants whereas not all variants are valid according the *structure condition*. The figure depicts two variants, one valid and one invalid. If the generated model structure contains the atomic entity nodes *A2, D, E, F, L*, it would be

valid because the *structure condition $p_1+p_2+1*p_3=3+3+1*3<12$* is true. The second model structure variant contains the atomic entity nodes *A2, D, E, F, L, L*. It is not valid because the *structure condition $p_1+p_2+2*p_3=3+3+2*3<12$* is false.
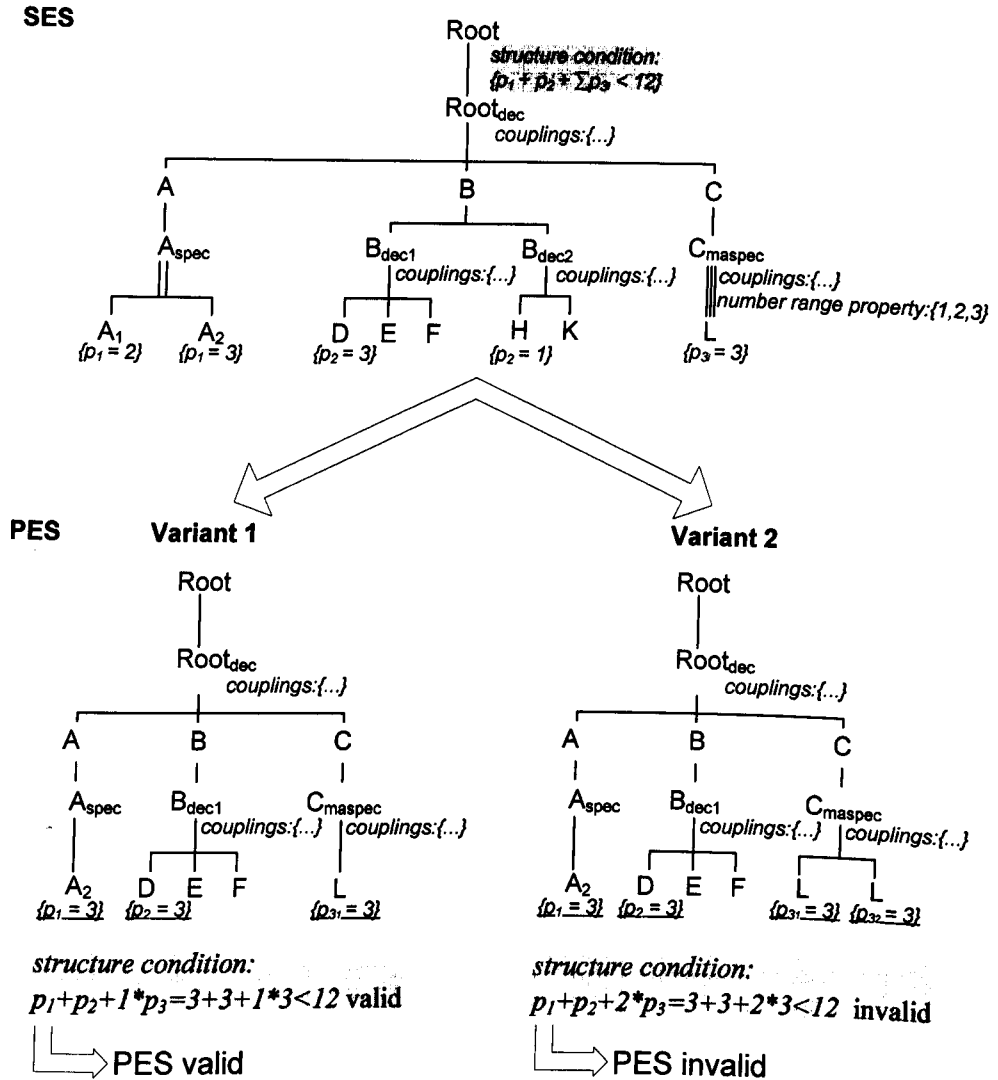
Figure 4.5 SES example with a structure condition

Chapter 5 provides the description of the application of the extended SES/MB framework. The chapter describes the combination of the introduced EDSDEVS formalism and SES/MB approach with an optimisation method to the simulation based parameter and structure optimisation as introduced in principle in section 2.3. The descriptions of the pruning and the terminal model generation processes, as a part of the SES/MB framework description, are provided in the context of other algorithms in chapter 5.

Chapter 5

A Framework for Modelling, Simulation and

Optimisation

Chapter 2 introduced the key research concept - simulation based parameter and structure optimisation as a merging framework of three methods, optimisation, model management, and modelling and simulation. Chapter 3 introduced EDSDEVS as a modular, hierarchical modelling and flexible simulation formalism as applied in the framework, and chapter 4 defines the SES/MB approach as a suitable model management framework. In this chapter a complete framework for combined parameter and structure optimisation experiments is proposed. After a brief description of the general framework structure, its methods are discussed in detail and the entire algorithm is summarised. Finally implementation details to describe a SES/MB structure with XML are introduced.

5.1 General Framework Structure

A fundamental overview of a simulation based parameter and structure optimisation experiment is shown in figure 2.5. A more detailed structure of the framework with concrete elements and information flow is depicted in figure 5.1. The interface definitions between the three modules are a fundamental part of this approach. They bind the named methods together to synthesise a simulation based parameter and structure optimisation.

On closer examination of the framework it is crucial to divide an optimisation experiment into two phases:

1. Initialisation phase

   The model management reads and analyses a meta-model. Results of the analysis are information about the multidimensional search space (X_S, X_P, D_S, D_P). The optimisation module is initialised with this information.

2. Optimisation phase

   During the optimisation phase the optimisation method explores the search space within a loop. Each examined search space point i.e. an ordered set of values (X_{S_i}, X_{P_i}) is delivered to the model management module. This module starts up the processes: structure synthesis, model generation, model simulation and performance estimation. The optimisation loop ends when a stop criterion is fulfilled. Examples of stop criteria are (i) going below a minimum alteration rate or (ii) exceeding the maximum number of optimisation cycles. The result of a successful finished optimisation phase is a parameter and structure optimised model.
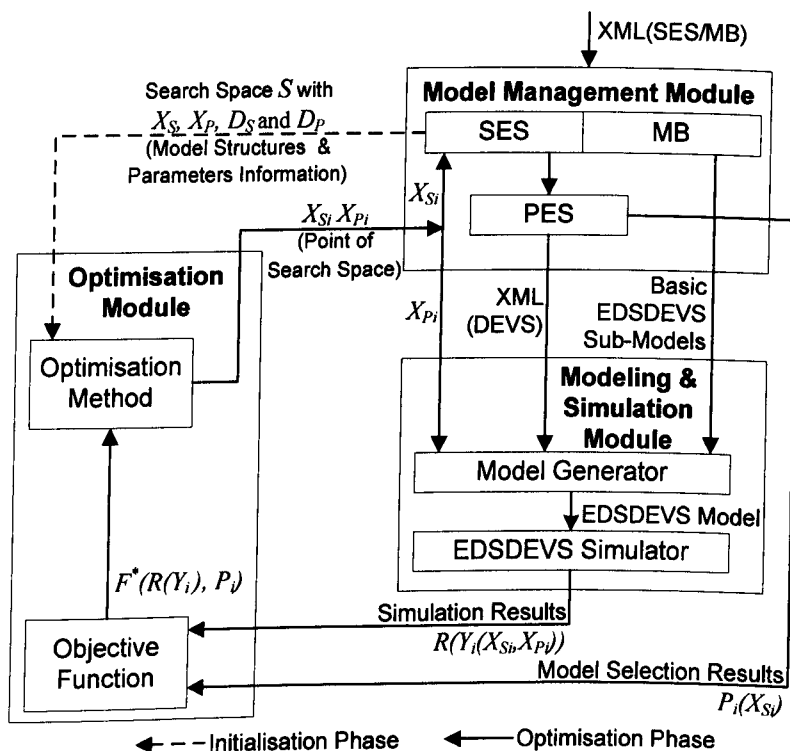
Figure 5.1 Structure of the simulation based optimisation framework

The simulation based optimisation framework is segmented into the following modules, methods and interfaces as depicted in figure 5.1:

1. Model Management Module: meta-model specification

   A meta-model definition is read and interpreted by the model management during the initialisation phase. A meta-model is defined in the form of a platform and implementation independent XML file. The basic components of a MB are regular EDSDEVS model components. They are referenced by the XML file with a model name and a model instance name. The result of this step is a data structure with an SES tree and references to a MB.

2. Interface Model Management Module – Optimisation Module: meta-model analysis

   In a second step during the initialisation phase the model management module analyses the SES tree and establishes the search space. The search space is defined by a set of variables with their domains. These sets X_S, D_S, X_P and D_P are sent to the optimisation module.

3. Interface Optimisation Module – Model Management Module: transformation of a search space points into a model configuration

   - The model management module receives a search space point (X_{Si}, X_{Pi}) within the optimisation loop. The sets X_{Si} and X_{Pi} are used to prune the SES, to synthesise the model structure and to parameterise the model. The selected model structure and model parameters are sent to a model generator as a platform and implementation independent XML files.

4. Model Generation Method

   Based on the received XML file with model structure information and references to basic components the model generator creates an EDSDEVS model.

5. Simulation Method

The EDSDEVS model is executed by an EDSDEVS simulator. In this research the modelling and simulation method is based on the EDSDEVS formalism. Principally this approach is not limited to EDSDEVS or DEVS formalisms exclusively.

6. Interface Model Management and Simulator – Objective Function

   In this approach the objective function gets both simulation results from the simulator and model structure selection results from the model management module to establish the performance of the current model structure and parameter set.

7. Optimisation Method

   The optimisation method establishes the next search space points to examine in a loop until the stop criterion is fulfilled. The search space points are chosen based on the search space definition and on previous objective function results.

5.2 Interface: Optimisation Module – Model Management Module

During the initialisation phase, the Model Management Module has to analyse the SES tree to transform formal meta-model structure information into numerical data useable by the Optimisation Module. Together with the model parameters the information is sent as initialisation data to the Optimisation Module. The information, coded in the four sets X_S, D_S, X_P and D_P is used to build the set $X^* = X_P \cup X_S$ and the corresponding domain set $D^* = D_P \cup D_S$. During the optimisation phase repeated in each optimisation loop cycle the optimisation method calculates a numerical data set $X_i^* = X_{Pi} \cup X_{Si}$. The set X_i^* is sent to the Model Management Module, which determines based on this information a new model configuration, i.e. a new model structure and initial model parameters. Both transformations are described by an example illustrated in figures 5.2 and 5.3.

The main task of the first transformation is to convert SES structure information to a structure parameter set X_S and the corresponding domain set D_S. This is done by a tree analysis starting at the root node, traversing the tree in a defined direction and considering every node. If a node is a decision node, i.e. it is a specialisation node, multiple aspect node

or composite entity node with alternative successor nodes, a structure parameter x_{Si} is added to the structure parameter set X_S and a corresponding domain d_{Si} to the domain set D_S. The domains of specialisation node and composite entity node are {1, ..., number of variants}. The domain of a multiple aspect node is defined by its attached number range property.

Two general principles can be applied to traverse the tree: (i) depth-first and (ii) breadth-first analysis. An advantage of the breadth-first analysis is the arrangement of the variables. If it can be assumed that variant decisions at a higher level of the SES tree have larger effects on the overall model structure than decisions near the leafs, a breadth-first analysis should be preferred. The breadth-first analysis sorts the elements of X_S and D_S as follows: elements on the left hand side of the ordered set correspond to higher levels of the SES; elements on the right hand side correspond to decision nodes nearer the leafs. An optimisation method could take this into account. Figure 5.2 illustrates the algorithm for creating structure parameter set X_S and the corresponding domain set D_S based on SES tree information. The analysis and X_S, D_S set build-up order is marked with small sequence numbers.
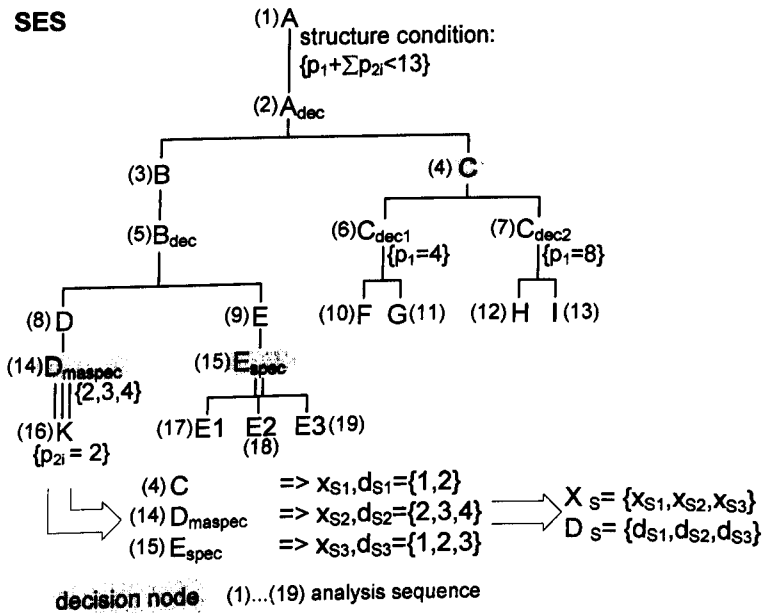
Figure 5.2 Transformation SES → set X_S and set D_S

The breadth-first analysis starts at the root node A, a non-decision node. Next nodes are non-decision nodes A_{dec} and B. The composite entity node C is the first decision node. It has two alternative successors. A first parameter x_{S1} is added to set X_S with the domain $d_{S1} = \{1, 2\}$. The next examined nodes are B_{dec}, C_{dec1}, C_{dec2}, D, E, F, G, H and I - they are non-decision nodes. The next examined node, the multiple aspect node D_{maspec} is a decision node. The value of its number range property is $\{2, 3, 4\}$. A second parameter x_{S2} is added to X_S with the domain $d_{S2} = \{2, 3, 4\}$. The next node, the specialisation node E_{spec} is again a decision node. It has three alternative successor nodes. A third parameter x_{S3} is added to X_S with the domain $d_{S3} = \{1, 2, 3\}$. The last nodes analysed K, E_1, E_2 and E_3 are non-decision nodes. The example SES has three decision nodes. The resulting structure parameter set is $X_S = \{x_{S1}, x_{S2}, x_{S3}\}$ with the corresponding domain set $D_S = \{d_{S1}, d_{S2}, d_{S3}\}$ with the above determined domains. On the basis of the combination of these sets X_S, D_S, the model parameter set X_P and its corresponding domain set D_P the optimisation method is able to search the search space. Additional SES tree information e.g. the structure condition at node A and the attached variables p_1 and p_2 in figure 5.2 are irrelevant during the initialisation phase.

- The second transformation is the reverse of the first. The Model Management Module receives a point in the search space from the Optimisation Module i.e. the numerical data set $X_i^* = X_{Pi} \cup X_{Si}$, where set X_{Si} codes a specific model structure and set X_{Pi} codes its model parameters. It has to synthesise the corresponding model structure and has to infer the model parameters. The transformation has to traverse the tree in the same direction as during the first in the initialisation phase. At each decision node the next element of current structure parameter set X_{Si} is used to decide: (i) which successor of a composite entity node with alternative successors nodes is chosen, (ii) which specialisation of a specialisation node is chosen or (iii) how many successors of a multiple aspect node are incorporated into the PES. After pruning the model structure is verified with the evaluation of all structure conditions. If a structure is invalid the specific set X_i^* will be refused and this information is

sent to the Optimisation Module. It marks this point in the search space as prohibited and determines a new one. Figure 5.3 illustrates the principle of this transformation. The analysis and pruning order is marked again with small sequence numbers.
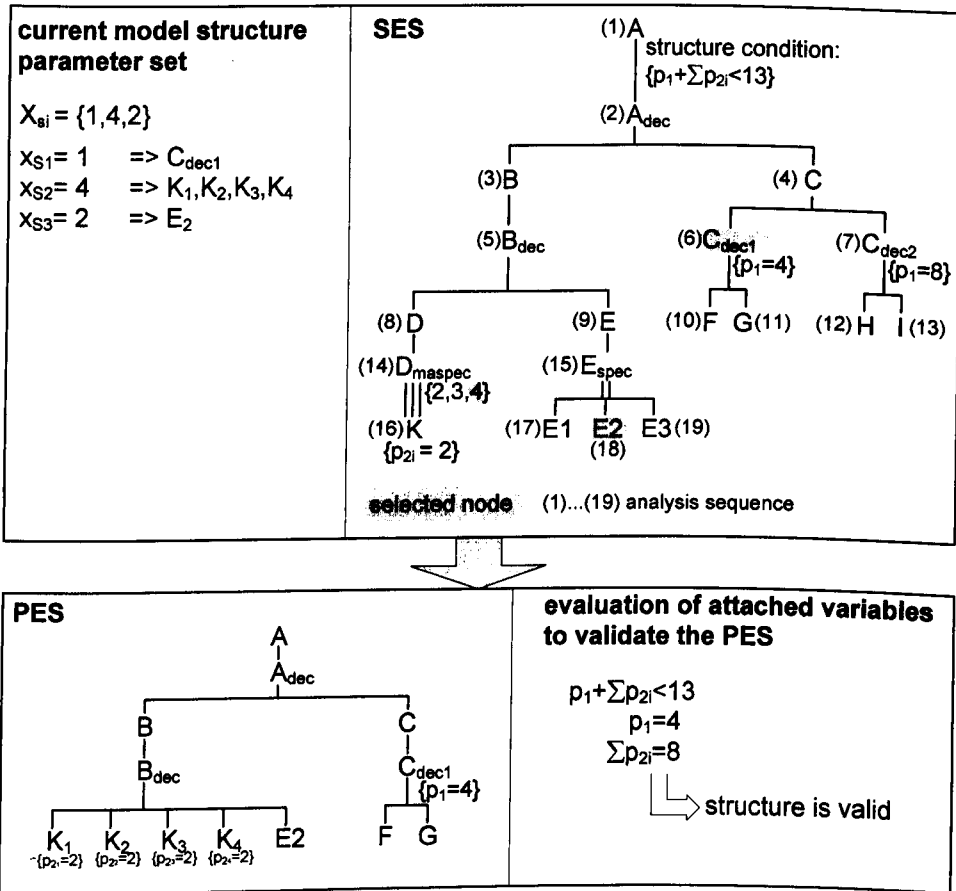
Figure 5.3 Transformation X_{Si} + SES $\rightarrow$ PES

The breadth-first analysis starts at the root node A and continues as already described before. The first decision node of the SES tree in figure 5.3 is composition entity node C. The first element in X_{Si} is $x_{S1}=1$, i.e. the first aspect node C_{dec1} is chosen for the PES. The next decision node is the multiple aspect node D_{maspec} and the corresponding set element is $x_{S2}=4$, i.e. the PES contains four nodes K. The last decision node is specialisation node E_{spec} and the corresponding set element is $x_{S3}=2$, i.e. the PES contains the second specialisation of node E_{spec}. After pruning, the attached variables are calculated and the PES is verified by evaluating the relevant structure conditions. In the example in figure 5.3, the aspect node C_{dec1} and four atomic entity nodes K were chosen. Therefore, the structure condition at node

[85]

A is evaluated as follows: $p_1 + \Sigma p_{2i} = 4 + 8 < 13$ and from this it follows that the PES is valid.

5.3 Interface: Model Management Module – Modelling and Simulation Module

Each optimisation cycle requires a change and adaptation of the simulation model. If the structure parameters in X_{Si} are changed, a new simulation model structure has to be generated. Otherwise, if just the model parameters in X_{Pi} are changed, it is adequate to re-initialise the model parameters. As illustrated in figure 5.1 all necessary information is sent from the Model Management Module to the Model Generator of the Modelling and Simulation Module. The Model Management Module creates XML files describing the model structure. EDSDEVS basic components, predefined in the MB, XML files and current model parameters coded in set X_{Pi} are used by the Model Generator to generate the entire EDSDEVS model.

The use of a standardised XML model description for information exchange decouples the two modules. It is based on W3C XML schema Finite Deterministic DEVS Models introduced in [30] and [31]. The XML interface uses the atomic and coupled model interface descriptions with model and port names. The coupled model description described in [31] is currently work in progress and does not contain all necessary description elements for this approach. Therefore, the composition description of coupled models additionally defines sub model names and coupling specification. The coupling specification defines external input (EIC), external output (EOC) and internal coupling information (IC). An example with corresponding XML files is illustrated in figure B.1 and listings B.17 and B.18 in appendix B.

The decoupling of Model Management Module and Modelling and Simulation Module using XML files eases the modelling and verification of the basic components

without the Model Management Module. Additionally it will enable and ease the use of different simulator implementations; however this will be the subject of future work.

5.4 Interface: Modelling and Simulation Module – Optimisation Module

The objective function, defined in the Optimisation Module, (figure 5.1), estimates the performance of the current model structure and parameter values. The function gets its input parameters from the Modelling and Simulation Module. These are the simulation results $Y_i(X_{Si}, X_{Pi})$ and simulation response function results $R(Y_i(X_{Si}, X_{Pi}))$ respectively. Further input parameters are delivered by the Model Management Module. These are the model structure results $P_i(X_{Si})$, which are based on evaluation of attached variables after pruning the SES. An example is illustrated in figure 5.2. The aspect nodes C_{dec1} and C_{dec2} and the atomic entity node K define the attached variables p_1 and p_{2i}. After the pruning process illustrated in figure 5.3 the values of p_1 and p_2 are calculated as follows: $P_i(X_{Si}) = \{p_1; \Sigma p_{2i}\} = \{4;8\}$. These values may be used as further objective function parameters.

The result $F^*(R(Y_i), P_i)$ of the objective function is evaluated by the optimisation method. As a consequence of the often stochastic nature of simulation problems, a random based optimisation method is preferable. Two established random based algorithms inspired by the principle of the evolution of life are the Genetic Algorithm (GA) introduced by Holland [20] and the Evolutionary Strategy (ES) introduced by Rechenberg [50]. The origins of ES are continuous parameter problems whereas current GAs support hybrid problems. There is an extensive and varied body of literature on this topic. Genetic algorithms have delivered robust solutions for various simulation based optimisation problems e.g. in [47] and [49]. Experiments realised within the scope of this research have shown that a GA is applicable as an optimisation method for the simulation based optimisation approach.

The methods of the simulation based parameter and structure optimisation framework described in this chapter are integrated into a general GA algorithm (listing B.19

in appendix B). The resulting algorithmic summary of the whole framework is introduced in the next section.

5.5 Algorithmic Summary of the Framework

As described in the preceding sections, the proposed simulation based parameter and structure optimisation framework is composed of different methods that form a uniform optimisation approach. The following algorithm, based on the general description in [54], summarises the fundamental operations using a GA as optimisation method.

Initialisation Phase:

0. Analyse the SES and establish $X^* = X_P \cup X_S$ and $D^* = D_P \cup D_S$

1. Initialise a population of individuals (generation 0) with different $X_i^* = X_{Pi} \cup X_{Si}$

Optimisation Phase (repeat until stop criterion is fulfilled):

2. Estimate the fitness of all individuals of the current generation

   Repeat for each individual

   2.1. Prune SES with X_{Si}

   2.2. If structure condition is valid, establish $P_i(X_{Si})$ or otherwise mark current individual as invalid and continue with next individual

   2.3. Generate EDSDEVS model

   2.4. Simulate EDSDEVS model and get result $Y_i(X_{Si}, X_{Pi})$

   2.5. Evaluate the simulation response function $R(Y_i(X_{Si}, X_{Pi}))$ by repeating step 2.4

   2.6. Evaluate the objective function $F^*(R(Y_i), P_i)$

3. Select pairs with m individuals and create descendants using crossover

4. Mutate the descendants

5. Exchange individuals of the current generation with descendants based on a substitution schema to create a new generation

A disadvantage of a conventional GA is the missing memory. It is possible that in different generations the same individual is repeatedly examined. Because of the time consuming

fitness estimation of one individual in simulation based optimisation, the addition of a memory method is vitally important. It has to store already examined individuals with their resulting $F^*(R(Y_i), P_i)$. This extension leads to the following, final algorithm summarising the fundamental operations of the simulation based parameter and structure optimisation approach using a GA as optimisation method:

Initialisation Phase:

    0.   Analyse the SES and establish $X^* = X_P \cup X_S$ and $D^* = D_P \cup D_S$

    1.   Initialise a population of individuals (generation 0) with different $X_i^* = X_{P_i} \cup X_{S_i}$

Optimisation Phase (repeat until stop criterion is fulfilled):

    2.   Estimate the fitness of all individuals of the current generation

        Repeat for each individual

        2.1. Check memory if current individual is known. In case of 'true': continue with next individual

        2.2. Prune SES with X_{Si}

        2.3. If structure condition is valid, establish $P_i(X_{Si})$ or otherwise mark current individual as invalid and continue with next individual

        2.4. Generate EDSDEVS model

        2.5. Simulate EDSDEVS model and get result $Y_i(X_{Si}, X_{Pi})$

        2.6. Evaluate the simulation response function $R(Y_i(X_{Si}, X_{Pi}))$ by repeating step 2.5

        2.7. Evaluate the objective function $F^*(R(Y_i), P_i)$

        2.8. Store X_i^* and $F^*(R(Y_i), P_i)$ in memory

    3.   Select pairs with m individuals and create descendants using crossover

    4.   Mutate the descendants

    5.   Exchange individuals of the current generation with descendants based on a substitution schema to create a new generation

5.6 Definition of a Model Set with XML SES/MB

In chapter 4 the extended SES/MB framework for the simulation based optimisation framework was formally introduced. This section describes the meta-model definition with the framework in detail. In this approach an SES/MB meta-model definition is based on XML [64]. Therewith the definition is platform and implementation independent. The usage of XML has the potential to enable the development of further extensions e.g. a graphical model designer. Figure 5.4 depicts the UML 2.0 [61] class and composition structure diagram of the XML schema and listing B.15 in appendix B contains the document type description (DTD [64]). Both the schema and the DTD are describing the structure of an SES/MB XML file.

The structure is divided into three main sub structures (i) SES tree, (ii) MB, (iii) properties:

1. The SES tree sub structure is defined within the *ses* sub tree of the XML structure. The six nodes (i) *composite*, (ii) *atomic*, (iii) *multiaspect*, (iv) *aspect*, (v) *specialisation* and (vi) *specialisation-entity* correspond to the different entity types of the formal SES/MB description as introduced in chapter 4. An exception is the specialisation-entity node which matches an atomic node. It is introduced to eases the SES XML file verification. The connections within the UML class and composition diagram defines the container class/contained class relationship and the m:n relations between both components. Each component has one attribute, the entity name *esname*. This name is used to logically connect XML elements within the XML SES, MB and property sub structures e.g. an *atomic* entity definition from the *ses* sub tree with the *mb_atomic* model implementation definition from the *modelbase* sub tree.

2. The MB is defined within the *modelbase* sub structure. The sub structure references (a) model implementations and defines (b) model interfaces:

   a. Nodes of the type *mb_atomic* and *mb_specializationentity* references basic components. The models are not directly defined within an SES/MB XML

file. The above nodes refer to a model implementation. The attribute *classname* refers to the model implementation class name and the attribute *modelname* names the specific model instance name. Both class and instance names are necessary to enable multiple usage of a component. The node *mb_aspect* is not a reference to a model implementation but is used to synthesise a model during model generation.

b. Nodes of the type *atomic, specialization* and *aspect* have attached coupling information. Hence the corresponding *modelbase* nodes *mb_atomic, mb_aspect* and *mb_specialization* define interfaces with input and output ports. Each model i.e. the corresponding structure in the *modelbase* can have several *inports* and *outports* named with the attribute *name* and combined in list structures *inports* and *outports*. Even though a specialisation node does not have a model implementation it has a definition in the *modelbase* sub tree. All child nodes of a specialisation share the same interface description which is defined once at parent node level.

3. To avoid scattered node property definitions all properties are defined in the *properties* sub structure. An aspect node defines a coupled model i.e. besides the sub components defined within the *ses* sub structure additional coupling information are necessary. A *modelcouplings* sub structure with a corresponding name in the *esname* attribute describes the coupling information in *eic, eoc* and *ic* lists. The number of possible children of a multiple aspect node is defined by the *varNumberOfComponent* structure. Nodes can have attached variables defined within the *var* structure and coupled with the *esname* attribute to the corresponding *ses* sub structure. Optional structure conditions are defined within the *constraint* structure.

SES sub-structure

ses_mb

ses

composite
-esname : string(idl)

aspect
-esname : string(idl)

specialization
-esname : string(idl)

atomic
-esname : string(idl)

specializationentity
-esname : string(idl)

multiAspect
-esname : string(idl)

Property sub-structure

properties

modelcouplings
-esname : string(idl)

eic
-inport : string(idl)
-component : string(idl)
-component_inport : string(idl)

eoc
-component : string(idl)
-component_output : string(idl)
-outport : string(idl)

ic
-component1 : string(idl)
-component1_outport : string(idl)
-component2 : string(idl)
-component2_inport : string(idl)

varNumberOfComponents
-esname : string(idl)
-min : unsigned long(idl)
-max : unsigned long(idl)

constraint
-name : string(idl)
-typ : string(idl) = {entity|parameter}
-action : string(idl) = {enable|valid}
-condition : string(idl) = {gt|lt|eq|gteq|lteq|neq}
-var_name1 : string(idl)
-var_name2 : string(idl)
-destination : string(idl)

var
-name : string(idl)
-esname : string(idl)
-typ : string(idl) = {internal|external}
-external_string : string(idl)
-value : string(idl)

MB sub-structure

modelbase

mb_specializationentity
-esname : string(idl)
-classname : string(idl)
-modelname : string(idl)

mb_atomic
-esname : string(idl)
-classname : string(idl)
-modelname : string(idl)

mb_specialization
-esname : string(idl)

mb_aspect
-esname : string(idl)
-classname : string(idl)
-modelname : string(idl)

inports

outports

inport
-name : string(idl)

outport
-name : string(idl)

Figure 5.4 UML Diagram of SES/MB XML Schema

The example SES in figure 5.5 defines two structure variants through two different specialisations A_1 and A_2 at A_{spec}. With the structure condition at the *ROOT* entity the PES can be verified after pruning. Figure 5.5 depicts the structure variants after pruning and model generation. Due to the structure condition only one model variant is valid. The listing B.16 in appendix B shows the corresponding meta-model definition with an SES/MB XML file. The three sub structures *ses*, *modelbase* and *properties* are separated with an empty line, XML elements, attributes and values are highlighted with different colours.
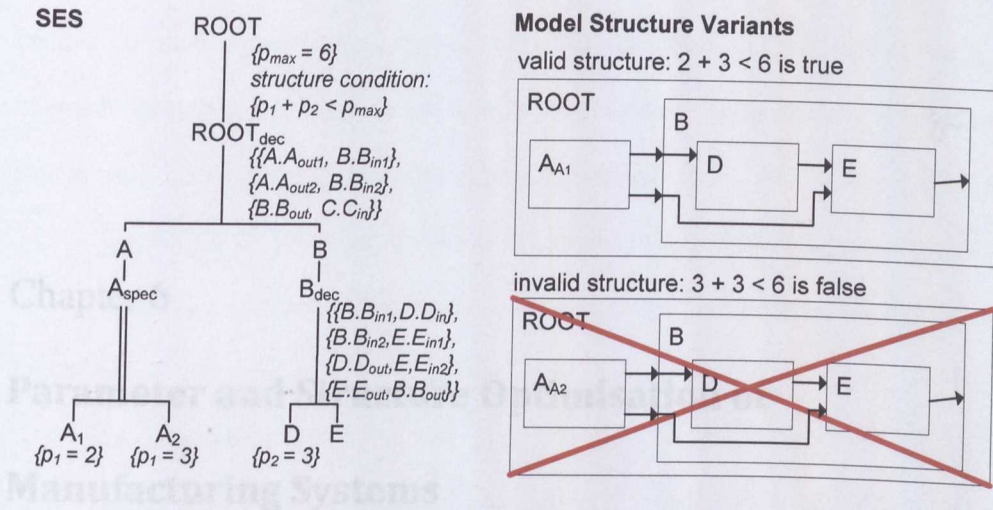
Figure 5.5 An SES/MB XML example – SES tree with both valid and invalid model structure variants

The next chapter starts with an overview of modelling and simulation of manufacturing systems and demonstrates the application of the introduced framework with a project from industry.

Chapter 6

Parameter and Structure Optimisation of Manufacturing Systems

This chapter demonstrates the application of the introduced framework for a simulation based parameter and structure optimisation with a real industrial project. It starts with a short review of types, components and complexity of manufacturing systems in the context of modelling and simulation. Current manufacturing system planning concepts and a range of modelling and simulation concepts for manufacturing system simulation are presented in an overview.

A broad choice of modelling and simulation packages is commercially available, developed to reflect the changing requirements of manufacturing applications. As discussed in chapter 2 not all demands of manufacturing modelling and simulation are satisfied optimally. A real life example using the approach developed in this research demonstrates how this can be accomplished.

6.1 Manufacturing Systems

The focus of manufacturing is the combination and transformation of raw material to a product with a market potential using industrial machines [21] [22]. This is a very simple principle but is difficult to achieve and maintain. The challenge is that the market potential and the requirements of manufacturing system are changing continuously. A manufacturer who does not adapt will lose competitiveness and vice versa a company that handles these

changes most effectively will succeed. A major issue for managers and engineers is the continuous analysis of manufacturing system performance and the use of methods to improve operations and adapt to new market situations. Analysis using modelling and simulation is potentially a powerful management method.

Depending on the point of view it is possible to differentiate between several types of manufacturing systems. Two widely used, described in more detail in [5] are the following:

- serial system

    An assembly line as a typical example of a *serial system* is a sequential set of workstations connected by material handling systems. Component parts are assembled or machined to produce a finished product in a line. The assembly activity can be divided into work elements as the smallest unit of productive work. A subset of work elements are assigned to each workstation. A work piece passes the complete line in a sequence. After leaving the final workstation the product is complete. Such systems are often used to produce a high volume of a small number of similar products. Figure 6.1 shows an example of a serial system with several lines with sub assembly manufacturing and a final end product assembly line.

Figure 6.1 General assembly system layout (source [5])

- shop scheduling system

  In contrast to a serial system a *job scheduling system* manufactures a variety of different products. Work pieces can follow different routes with significant different processing time at a workstation. Regularly work pieces are combined in batches or jobs of one or more parts which are manufactured on the same route i.e. with the same production sequence and similar processing time. If all batches are processed in the same sequence of workstations the system is named *flow shop*. In contrast, in a *job shop* each batch type has the same production sequence. With a growing flexibility and pressure to decrease manufacturing cost the complexity of job shop systems is increasing considerably. Hence the planning of *job shop* systems is making greater than ever demands.

6.2   Modelling and Simulation of Manufacturing Systems

The simulation of manufacturing and material handling systems is one of the most important applications of discrete event modelling and simulation techniques [7]. These techniques have been successfully used as an aid in the design of new systems as well as an evaluation tool for improvements to existing systems, as a daily staffing, material and operation planning tool and so on.

Even though both the types of manufacturing systems and the analysis issues vary substantially the different modelling and simulation techniques share some common characteristics as described in the following sections.

6.2.1   Simulation Model Level of Detail

In principle every model is an approximation of the real world. Depending on the analysis objectives irrelevant characteristics and details can be omitted when creating a model. In simulation literature this principle is called *level of abstraction* [51] because the model is an abstraction or approximation of the real system. The appropriate level of detail can

distinguish between valid and invalid or successful and unsuccessful simulation experiments. It is claimed that a good rule is to add details step by step during a model validation process because starting with a low level of detail usually leads to fewer simulation results to be validated [51]. The analyst stops the process when the model is close enough to real system behaviour to provide results for analysis. This validation approach is an iterative process that results in a sufficiently accurate model. Figure 6.2 depicts the correlation between model detail and validation time [51]. The asymptotic behaviour of the relationship may mean more effort to increase the level of detail from 95% to 100% than creation of the initial model with 95% accuracy.

Figure 6.2 Model detail during model validation (source [51])

6.2.2 Fundamental Components

Manufacturing systems produce a wide range of products with many types of production methods using many different system layouts. Nevertheless there are common components that can describe many manufacturing operations. These common components are the basis elements of a simulation model [51]. Table 6.1 depicts these basic elements.

Product	Resource	Demand	Control
Parts/pieces	Equipment layout	Customer orders	Warehouse management
Routings	Equipment costs	Start date	Inventory control
Process time	Number of machines	Due date	Shop floor control
Setup time	Failure	WIP inventory	WIP tracking
Bill of material	Maintenance		Restricted resources
Yield	Number of operators		Station rules
Rework	Shift schedules		

Table 6.1 Fundamental components of manufacturing systems (source [51])

Product. *Parts or pieces* are the products manufactured. Products may be handled as a single item or production unit or combined to batches depending on the manufacturing process named batch or job. A batch can be described as a production unit in a subsequent process. Products are manufactured in a defined sequence, the *routings*. Depending on the manufacturing process and on the product the *routing* can be sequential e.g. in an automobile assembly line i.e. a serial system or complex e.g. in a semiconductor production process i.e. a job scheduling system. For each manufacturing step the *setup and processing time* determine the total cycle time. These times depend on the machine and/or product and can be deterministic or stochastic.

A product can be assembled from several items, i.e. sub assemblies, defined by the product structure file or *bill of material (BoM)*. Each item in the BoM can be the result of a production process. During the manufacturing process all BoM items must be available at a defined point of time relative to the final product assembly or product due date. The modelling of manufacturing systems with a delivery or production of sub assemblies Just-In-Time to minimise waste and inventory is an important manufacturing paradigm today. The typical example of this principle is the automobile industry.

Yield and *rework* are found in many manufacturing processes. The reasons are imperfect processes and operations. Both factors influence the process throughput and other

characteristics e.g. the costs. With a lower level of detail both characteristics can also be omitted.

Resources. Resources include machines and human operators, mobile and immobile equipment, material and storage systems etc. They are used to manufacture a product. The *equipment layout* and the *number of machines* have an effect on the production flow and the speed of operation. The *equipment costs* influence amongst others the manufacturing cost of a product. Staff number can be a restricted resource, e.g. the number of machines and with these the necessary *number of operators* is higher than the available number of operators. In this context *shift schedules* have to be possibly considered.

The equipment has unplanned and planned down times, random *failures* or regular *maintenance*. During these times production has to stop or product flow has to be rerouted when alternatives are available.

Demand. *Costumer orders* define the demands on a manufacturing process. *Start and due dates* are determined by these customer orders for products. An important question of production management is the determination of the latest *start date* for BoM items to complete the order before the *due date*.

Normally production does not start from an idle state instead there is some work-in-process (WIP) e.g. in buffers, on conveyors or in machines. The modeller can decide to accept an initialisation phase until the model contains a certain amount of WIP to start the real experiment or initialise the model with work-in-process data.

Control. Control systems determine how products flow through the manufacturing processes, collect status information about products and/or resources, inspect the compliance of resource or demand constraints and decide about the use of the *restricted recourses*. A control algorithm can influence a simulation with changes of input data e.g. a changed

semi finished part order in an assembly line or changes in inwards and outwards goods movements in a *warehouse management* system. A *shop floor* or/and an *inventory control* algorithm can change model properties and model structure e.g. a storage area extension or reduction or an equipment layout modification of a manufacturing system. A *WIP tracking* system can deliver current process status information for control strategies. *Station rules* define local scheduling decisions, e.g. the working sequence in a manufacturing cell from simple first in, first out control strategy to a more complex such as a custom order dependant priority control strategy.

6.2.3 Measures of Performance

The methods to evaluate the performance of a real system and model have to be the same otherwise it will be difficult to have confidence in simulation and analysis results. Because both the real system and its model are based on random events the performance measure is a statistical analysis of real system and simulation system results. The following measurements are typical for a manufacturing system [51]:

- Throughput of sub model (such as a machine or process) or the complete model

- Cycle time at a process or overall

- Queueing time or length

- Response time of material handling equipment

- WIP

- Resource utilisation

- System specific performance measures (scrap rate, waiting time at a process etc.)

Due to the fact that a manufacturing system is a complex system it is important to note that model changes to improve one measure usually change other measures, for optimisation this is an important issue.

6.2.4 Analysis Issues

Using the measures described in chapter 5 an analyst experiments with a model to understand coherences of model elements and the behaviour of the whole system using input value, model parameter and model structure changes. Among others the following are typical analysis questions [51]:

- Determining bottlenecks

- Determining required staffing levels

- Evaluating the scheduling of staff

- Evaluating the scheduling of tasks

- Evaluating the control system

- Recovering strategies for random events

The identification of bottlenecks is often an analysis issue. The problem is the direct influence of the experiment on the bottleneck. With changes of anything in the model the primary bottleneck can move to other elements of the model. So the identification of a bottleneck can be a complex task and requires the examination at both local and global model levels.

A second important analysis issue is the determination of resource levels. Manufacturing systems with a fluctuating production volume, e.g. with seasonal dependencies, require such an analysis. An example is the staff requirement. It can change constantly and has to be planned regularly. An associated issue is the scheduling of staff between manufacturing system elements. With intelligent scheduling strategies it may be possible to employ fewer staff and still maintain sufficient throughput or to increase the throughput without increasing staff costs.

6.3 Introduction to the Photofinishing Industry

The application in this research uses developments and problems in the photofinishing industry and investigates a small part of a production process to validate the key research

concept. The photofinishing industry specialises in high volume production of thousands to millions of pictures per day but has nevertheless a relatively broad range of different products. As a consequence of significant changes in the photography market, notably the introduction of digital cameras with a considerable reduction of analogue and an increase of digital orders during recent years, a mix of analogue and digital production facilities are used. The change of the main production material from analogue to digital material has lead to concentration from many, local working, smaller laboratories to few, large, nationwide working laboratories and fierce competition between them. The situation is driving an urgent need to be as cost effective as possible.

Figure 6.3 shows general structure and product flow through the different departments of a typical photofinishing laboratory. It is possible to differentiate between three main production departments to depict the production flow analogue film/digital image – photographic picture:

I. The material arrives in several forms at the login process. After sorting the product mixes, some 10 to some 100 single orders are combined into batches. Each batch contains only one production material and one product type, e.g. undeveloped analogue film and specific paper width and surface. The batch creation is done with different machine types: (i) a splicer combines undeveloped film rolls onto a large film reel, (ii) a universal reorder station (URS) combines analogue reorders to a strap of film strips, (iii) a digital URS scans the analogue reorders and creates a digital batch, (iv) a digital splicer handles digital data carriers (CDs, flash cards etc.) and (v) software applications combine digital images collected by a web server. Steps (i) and (ii) creates analogue and steps (iii)...(v) digital batches.

II. Undeveloped analogue batches have to be developed. Analogue material can be scanned for the next steps which could be CD production and digital printing. As an alternative, the analogue batches are printed at an analogue printer. The result of both printer types is a huge reel of exposed photographic paper.

III.   After the development of a photographic paper reel the final step is cutting. Regarding paper cutting both cutter and digital cutter are comparable. A DigiCutter is specialised for paper cutting without a film cutter but possibly equipped with several paper cutters each able to cut different paper widths. Finally items are packed and identified for delivery to customers.
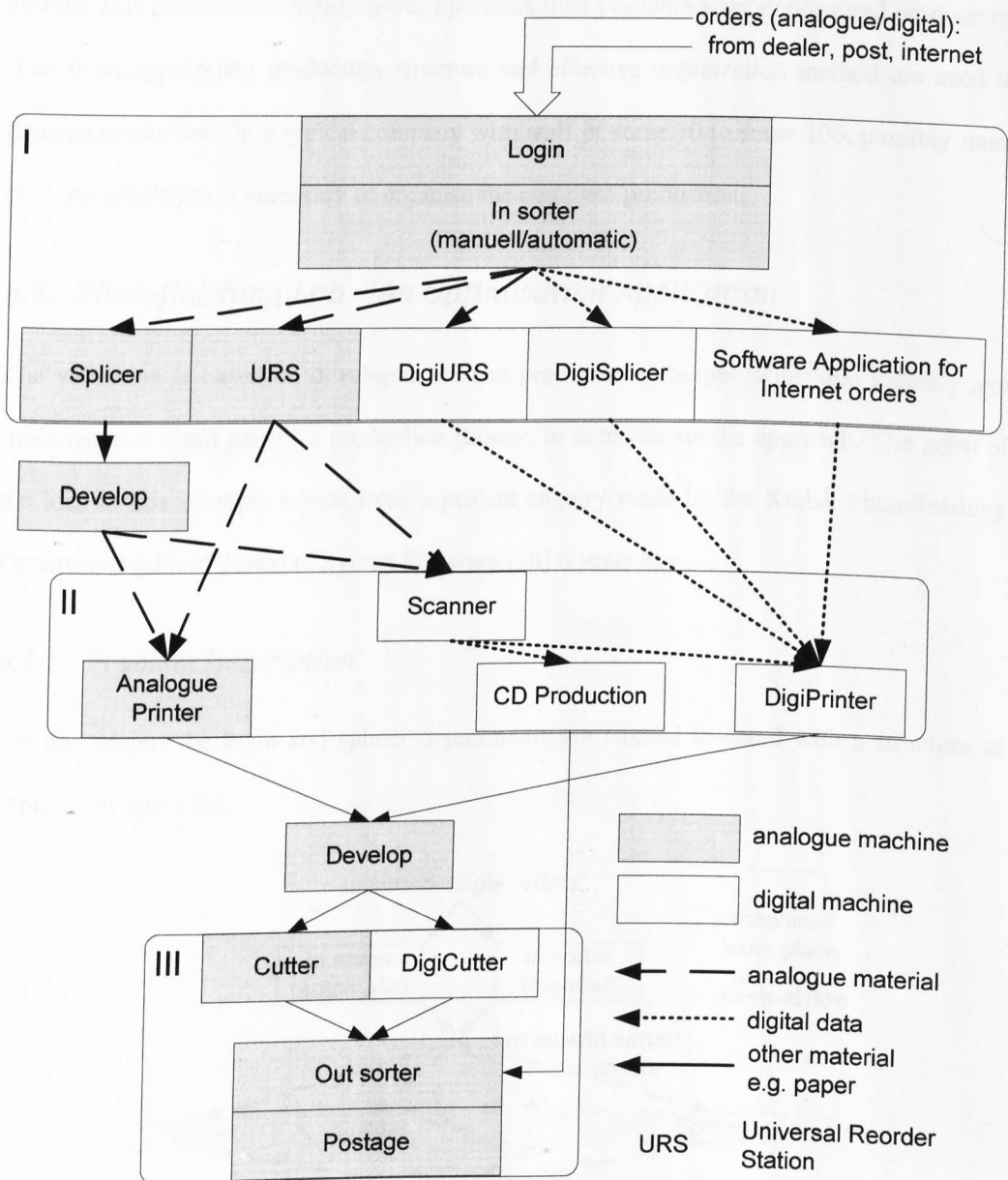
Figure 6.3 General product flows of a photofinishing lab

Figures C.1 ... C.4 in appendix C show a selection of photofinishing machines.

The product flow splicer/URS – development – analogue printer – development – cutter was the common production flow before the digital era and is typical serial manufacturing system. Nowadays there are several possible material routes through production with the same end product but different processing time, machine and operator requirements and costs i.e. a photofinishing lab now appears more as a job scheduling system. It is possible to employ fewer operators than available workstations and produce on time if an appropriate production structure and effective organization method are used to manage production. In a typical company with staff of some 10 to some 100, possibly more than one employee is necessary to organise the complete production.

6.4   Photofinishing Lab – An Optimisation Application

The validation is based on developments and problems in the photofinishing industry and investigates a small part of a production process to demonstrate the approach. The germ of the idea to this example comes from a project enquiry made by the Kodak Photofinishing Department (closed down) to Syntax Software [58] 6 years ago.

6.4.1   Problem Description

For this project the login and splicer departments are studied in detail with a structure as depicted in figure 6.4.
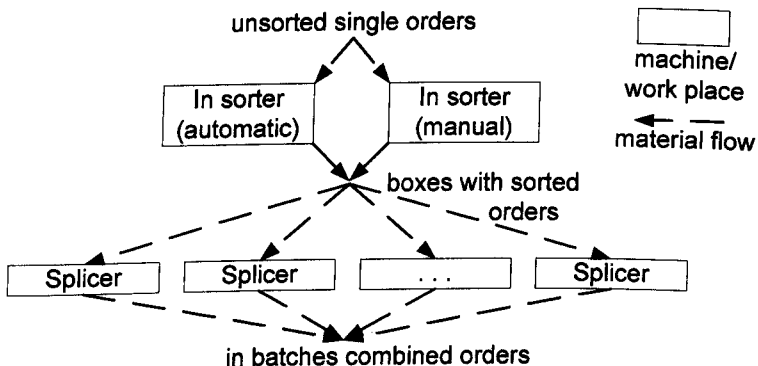
Figure 6.4 Product flow of the considered example

- System description

The source materials, unsorted, single orders, are sorted by product type manually or automatically into boxes. These sorted orders are combined to batch reels at splicers. An automatic sorter is handled by one or two operators, whereas manual sorting is done by the number of available operators without the need of a machine. A splicer is handled by one operator. Operators can be moved between machines. The handling time of the machines is listed in table 6.2.

Machine	Order handling time (s)
automatic in sorter	0.5
manual in sorter	$1.7 \leq 2$ (equal distribution)
splicer	$0.9 \leq 1$ (equal distribution)

Table 6.2 Order handling times

Sorting and splicing of a defined amount of orders takes a production time depending on type of machines, number of operators and organisation strategies. The production time is estimated by simulation.

A specific production system causes costs. In this case study the costs depends on the number of operators as shown in table 6.3.

# of operators	Costs
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

Table 6.3 Production costs

- Simulation model level of detail and fundamental components

Each workstation is taken as a black box with a defined processing time and resource utilisation. Workstations need a specific number of operators to manufacture and can be enabled or disabled. Further properties do not exit.

Source material is modelled as a data structure with material type and planed end product type.

A production or department manager is modelled as a control model. The model can enable and disable workstations and organise material flow depending on the availability of operators, unhandled source material and queue lengths.

The number of operators is a model property used by the control model. Operators are moved between departments and workstations to enable and disable workstations. Operator movements does not cost any time.

To minimise complexity additional considerations e.g. setup time, maintenance and failure are not modelled. Shift schedules and other components in connection with operators are not modelled too.

- Performance Measurement

For a performance measurement the sorting and splicing of a defined number of orders are simulated. The simulation output of a single run delivers the production time and cost $Y = \{y_{production\ time}, y_{costs}\}$ of the current model variant.

The simulation response function calculates the average over 50 runs. They are passed to the objective function that is defined by the term:

$$F = F(Y) = \alpha_1 * r_1 * \bar{y}_{production\ time} + \alpha_2 * r_2 * \bar{y}_{costs} \rightarrow minimum$$

The factors α_1 and α_2 normalise the values of the variables, $\bar{y}_{production\ time}$ and $\bar{y}_{costs}$. The factors r_1 and r_2 define the relevance of the variables, $\bar{y}_{production\ time}$ and $\bar{y}_{costs}$. With the factors $\alpha_1=1/max\_production\_time$, $\alpha_2=1/max\_costs$, $r_1=1$ and $r_2=1$ both variables are within the range between 0 and 1 and have the same relevance. The maximal value of the production time can be calculated with a minimal production system i.e. one operator, manual sorting and one splicer. The maximal value of the costs is defined by the upper bound of the parameter *number of operators*. In this case study for both variables, $\bar{y}_{production\ time}$ and $\bar{y}_{costs}$ the same relevance is chosen. Depending on the analysis objectives a different relevance of $\bar{y}_{production\ time}$ and $\bar{y}_{costs}$ can be used.

The result of the funtion F is the performance of the investigated model variant.

- Analysis issues

The production time and consequently the cost for a specific number of orders varies depending on the type and number of machines used, number of operators and the strategy to organise operators i.e. the initial distribution and succeeding movement of operators between machines and departments. The challenge for modelling is to minimise the combination of the production time of a given number of orders and the costs i.e. employing a minimal number of operators.

6.4.2 Implementation Details

Figure 6.5 shows the SES, describing possible model structures and the model parameter *number of operators*. Both the SES and the model parameter are the open quantities of the optimisation problem. The model structure variants are characterised by the use of: (i) automatic and/or manual sorting, (ii) one to eight splicers and (iii) one of three different department organisation strategies to share operators between machines and departments. Depending on selected alternative nodes during the pruning process several structure related attached variables will be initialised with different values. The SES defines 72 model structure variants in all. In addition there is one model parameter, the *number of operators* with a range of one to eight. The combination results in 576 model variants. Not all model variants define useful combinations. For example a model with four operators and eight splicers delivers the same result as a model with four operators and four splicers since in both variants only four splicers at all can be used. To exclude the useless variants the root node MODEL defines a structure condition that reduces the valid number of model variants to 275.

The following list describes the nodes and basic components, respectively:

- DEP_LOGIN

  The login department model can have three different sorting configurations. The first configuration applies only manual, the second only automatic and the third combines both sorting types. The number of available operators in this department is managed

by the controller$_{spec}$ model. Decisions of the controller$_{spec}$ model may be a function of the queue_order length.

- DEP_SPLICER

  The splicer department model can consist of a different number of splicers. The number of available operators in this department is managed by the controller$_{spec}$ model. Decisions of the controller$_{spec}$ model may be a function of the queue_box2 length.

- controller$_{spec}$

  The specialisation node controller$_{spec}$ has three successor nodes each implementing another staff organisation strategy:

  - ctrl1:

    The strategy starts with employing operators in the login department. If more staff is available than needed they are employed in the splicer department. After sorting is finished all staff is employed in the splicer department.

  - ctrl2:

    The strategy starts with employing operators in the login department. If more staff is available than needed they are employed in the splicer department. If the queue_box length is larger or equal than four all staff is employed in the splicer department. If the queue_box length is smaller than four the initial staff arrangement is recovered.

  - ctrl3:

    The strategy starts with employing half of operators in the login department and the other half in the splicer department. After sorting is finished all staff is employed in the splicer department.

Model Parameter   **SES**

#_of_operators={1,8}

MODEL

structure conditions:
{max(manu_login+auto_login,#_of_splicers)<=#_of_operators
AND
(manu_login*8+auto_login*2+#_of_splicers)>=#_of_operators}

MODEL$_{dec}$

DEP_LOGIN   CONTROLLER   DEP_SPLICER

controller$_{spec}$ DEP_SPLICER$_{dec}$

DEP_LOGIN$_{dec1}$ DEP_LOGIN$_{dec2}$ DEP_LOGIN$_{dec3}$

ctrl$_1$ ctrl$_2$ ctrl$_3$ queue_box2   queue_batch

splicer$_{maspec}$

queue_order   queue_box1   queue_order   sorter_manu {manu_login=1}   {#_of_splicers ={1,...,8}}

sorter_manu {manu_login=1}   sorter_auto {auto_login=1}   queue_box1   splicer

queue_order   queue_box1

sorter_auto {auto_login=1}
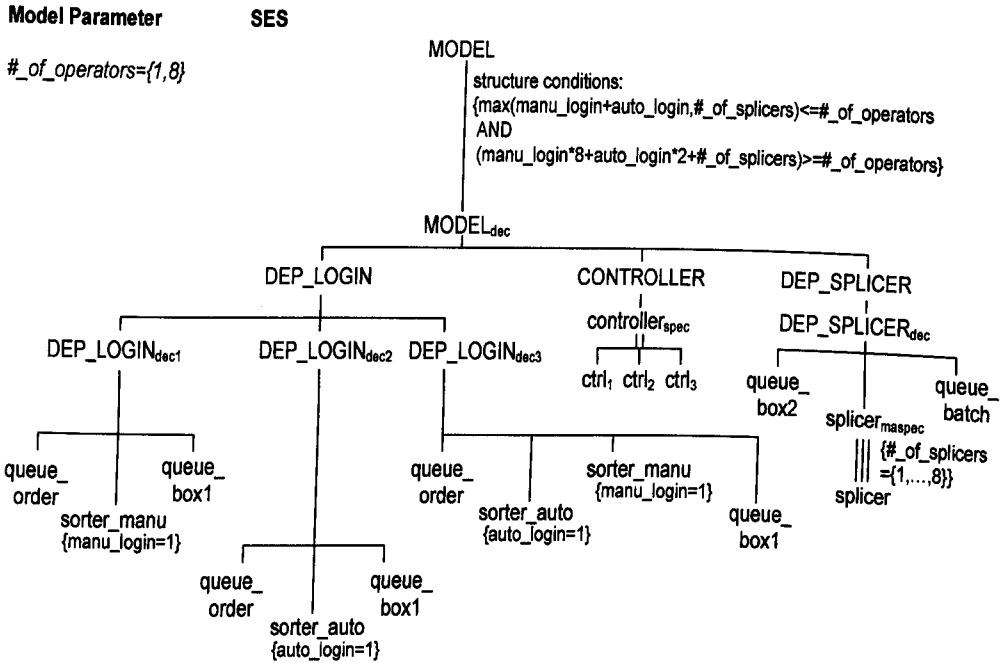
Figure 6.5 Model parameter and SES of the application

To solve this example, the search space has to be defined in terms of a structure parameter set, a model parameter set and their corresponding domain sets. Using the principle introduced in section 5.2 the structure parameter set and the corresponding domain set are defined by:

$$X_S = \{x_{DEP\_LOGIN}, x_{controllerspec}, x_{splicermaspec}\}$$

$$D_S = \{d_{DEP\_LOGIN}, d_{controllerspec}, d_{splicermaspec}\} \text{ with}$$

$$d_{DEP\_LOGIN} = \{1; 2; 3\}$$

$$d_{controllerspec} = \{1; 2; 3\}$$

$$d_{splicermaspec} = \{1; 2; 3; 4; 5; 6; 7; 8\}$$

The model parameter set and the corresponding domain set are defined by:

$$X_P = \{x_{\#\_of\_operators}\}$$

$$D_P = \{d_{\#\_of\_operators}\} \text{ with } d_{\#\_of\_operators} = \{1; 2; 3; 4; 5; 6; 7; 8\}$$

Hence, the resulting search space is defined by:

$$X = X_P \cup X_S$$

$$X = \{x_{DEP\_LOGIN}, x_{controllerspec}, x_{splicermaspec}, x_{\#\_of\_operators}\}$$

Each model variant defines one point in the search space. With the principle introduced in section 5.2 a PES can be derived and a corresponding model can be generated. One point of the search space is $X_{132} = \{2; 2; 2; 2\}$. This means that the aspect node DEP_LOGIN_{dec2} and the specialisation $ctrl_2$ are chosen, the number range property value of the multiple aspect node $splicer_{maspec}$ is two and the model parameter *#_of_operators* is also two. Figure 6.6 depicts the PES of model variant 132. The generated EDSDEVS model is illustrated in figure 6.7.
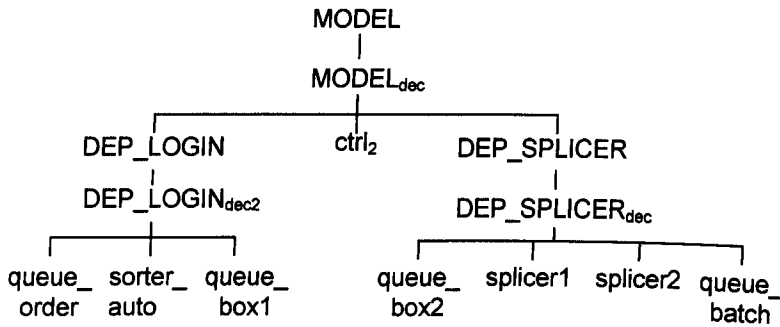
Figure 6.6 PES of 132th variant
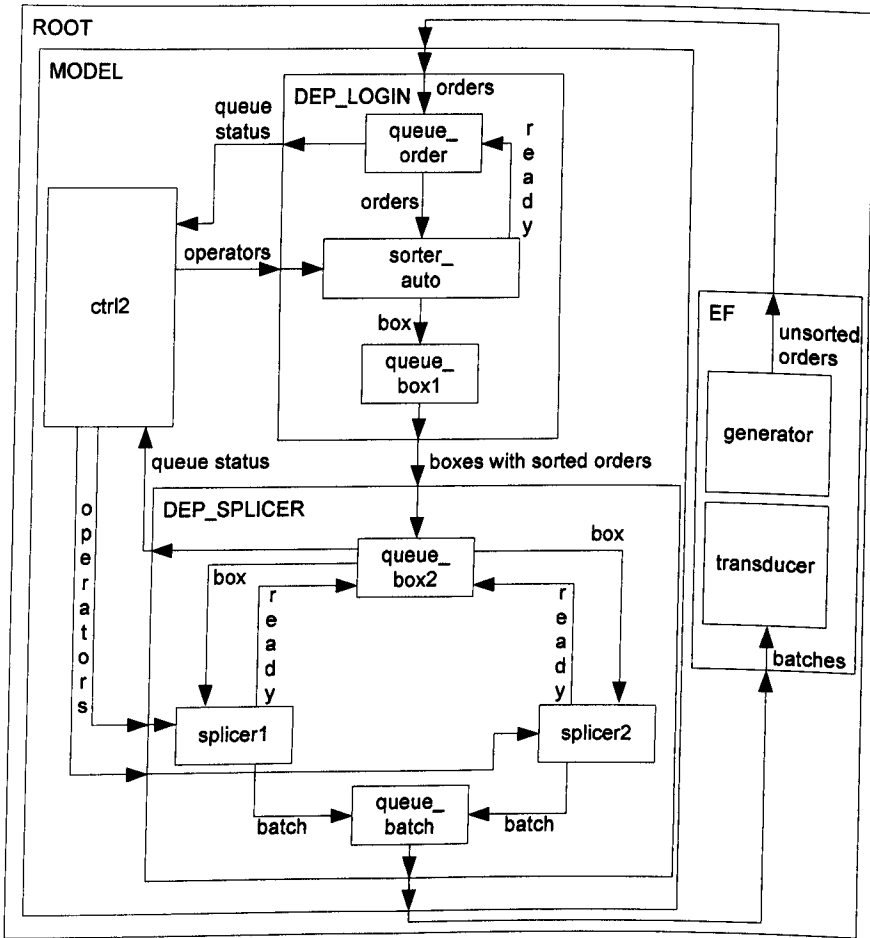
Figure 6.7 Model structure of 132th variant

All model variants use intensively the dynamic structure characteristics of the EDSDEVS formalism. The model of the department manager (model *ctrl2* in figure 6.7) activates and deactivates several atomic models (models *sorter_auto*, *splicer1* and *splicer2* in figure 6.7) and creates and destroys couplings respectively based on the department manager algorithm and the current model state. Figure 6.8 shows a sequence diagram section of one simulation run. Depending on queue lengths messages are generated and sent to the control model that enables/disables models and creates/destroys couplings.
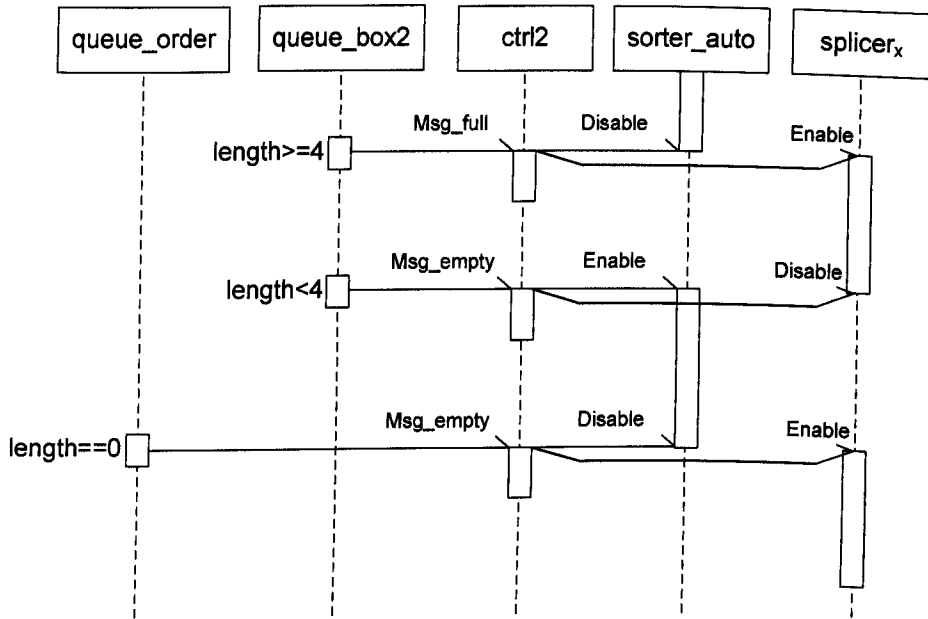
Figure 6.8 A sequence diagram section of one simulation run

Numerous commercial and non-commercial GA implementations exist. In this research the commercial toolbox MATLAB® GA toolbox [59] released by The MathWorks™ is used. The default MATLAB GA parameter settings were used, except for a decreased population size of 15 and an adjusted stop criterion:

if the weighted average change in the fitness function value over x generations (x=20 in 1st and x=5 in 2nd experiment) is less than 0.01, the algorithm stops.

In the following all GA parameters and their values are listed. A description and lists of possible values as well as the algorithm description can be found in [59].

Population:

- Population type: Double Vector

- Population size: 15

- Creation function: Uniform

- Initial population: []

- Initial scores: []

- Initial range: [0; 1]

Fitness scaling:

- Scaling function: Rank

Selection:

- Selection function::   Stochastic uniform

Reproduction:

- Elite count: 2

- Crossover fraction: 0.8

Mutation:

- Mutation function: Gaussian

- Scale: 1.0

- Shrink: 1.0

Crossover:

- Crossover function: Scattered

Migration:

- Direction: Forward

- Fraction: 0.2

- Interval: 20

Algorithm settings:

- Initial penalty: 10

- Penalty factor: 100

Hybrid function:

- Hybrid function: None

Stopping criteria:

- Generations: 100

- Time limit: Inf

- Fitness limit: -Inf

- Stall generations: 20 (1st experiment)

                                         5 (2nd experiment)

- Stall time limit: 20

- Function tolerance: 0.01

- Nonlinear constraint tolerance: 0.000001

Display to command window:

- Level of display: Final

Vectorize:

- Fitness function is vectorized: Off

The population size and the stop criteria are adapted for this case study. It is possible that changes of other parameters would lead to better optimisation results but further experiments are not undertaken in the scope of this research.

Each simulation run estimates the production time of 200 orders with a random production type mixture. The optimisation was repeated 50 times for each stop criterion with different random number generator initialisations. Listing 6.1 shows a Matlab code section of the optimisation initialising and executing the GA.

```
% ses tree is initialised outside of this function
function example_optim_exp(ses)

    % function uses two parameters, the ses object (global
    variable) and a search room point
    fitnessFunction = @exec_simu;

    % Bounds
    % e.g. LB = [ 1 1 1 1 ];
    % e.g. UB = [ 3 3 8 8 ];
    [LB UB] = ses.generateBounds();

    % Number of Variables
    nvars = size(LB,2);

    % Start with default options
    options = gaoptimset;

    % Modify some parameters
    options = gaoptimset(options,'PopulationSize' ,15);
    options = gaoptimset(options,'StallGenLimit' ,20); %1st exp.
    % options = gaoptimset(options,'StallGenLimit' ,5);%2nd exp.
    options = gaoptimset(options,'TolFun' ,0.01);

    % Run GA
    [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] =
```

```
ga(fitnessFunction,nvars,Aineq,Bineq,Aeq,Beq,LB,UB,nonlconFu
nction,options);
```

Listing 6.1 Matlab code section with GA initialisation and execution

6.4.3 Results

To validate the research framework the global optimum estimated through simulation of all system variants is compared with the result of an optimisation experiment. In both experiments the performance rating of a variant is established by the same objective function using the following function definition:

$$F = F(Y) = \alpha_1 * r_1 * \bar{y}_{production\ time} + \alpha_2 * r_2 * \bar{y}_{costs}$$

$r_1 = r_2 = 1$ – same relevance of both paramters

$\alpha_1 = 1/566$ – maximal production time with a minimal production system is 566 (1st line

in table 6.4)

$\alpha_2 = 1/8$ – maximal costs are 8

The simulation results of all 275 variants are shown in table 6.4. The columns control strategy, login and # of splicers specifies the model structure and the column # of operators specifies the model parameter as described in subsection 6.4.2. The production time values are the simulation result of the production of 200 orders. The costs correspond to the number of operators and the fitness is calculated with the above objective function.

ctrl strat.	login	# of splicer	# of ops	prod. time	costs	fitness
1	1	1	1	566,0	1	1,1250
1	1	1	2	357,0	2	0,8807
1	1	1	3	209,0	3	0,7443
1	1	1	4	208,0	4	0,8675
1	1	1	5	209,0	5	0,9943
1	1	1	6	208,0	6	1,1175
1	1	1	7	208,0	7	1,2425
1	1	1	8	207,0	8	1,3657
1	1	2	2	288,0	2	0,7588
1	1	2	3	208,0	3	0,7425
1	1	2	4	208,0	4	0,8675
1	1	2	5	207,0	5	0,9907
1	1	2	6	208,0	6	1,1175
1	1	2	7	208,0	7	1,2443
1	1	2	8	207,0	8	1,3657
1	1	3	3	209,0	3	0,7443
1	1	3	4	199,0	4	0,8516
1	1	3	5	208,0	5	0,9925
1	1	3	6	208,0	6	1,1175
1	1	3	7	208,0	7	1,2425
1	1	3	8	209,0	8	1,3693
1	1	4	4	208,0	4	0,8675
1	1	4	5	207,0	5	0,9907
1	1	4	6	208,0	6	1,1175
1	1	4	7	208,0	7	1,2425
1	1	4	8	208,0	8	1,3675
1	1	5	5	208,0	5	0,9925
1	1	5	6	197,0	6	1,0981
1	1	5	7	208,0	7	1,2425
1	1	5	8	208,0	8	1,3675
1	1	6	6	209,0	6	1,1193
1	1	6	7	208,0	7	1,2425
1	1	6	8	208,0	8	1,3675
1	1	7	7	198,0	7	1,2248
1	1	7	8	200,0	8	1,3534
1	1	8	8	208,0	8	1,3675
1	2	1	1	279,5	1	0,6188
1	2	1	2	229,5	2	0,6555
1	2	2	3	179,8	3	0,6926
1	2	2	2	139,75	2	0,4969
1	2	2	3	119,5	3	0,5861
1	2	2	4	99,5	4	0,6758

ctrl strat.	login	# of splicer	# of ops	prod. time	costs	fitness
1	2	3	3	99,8	3	0,5512
1	2	3	4	89,5	4	0,6581
1	2	3	5	69,3	5	0,7474
1	2	4	4	79,3	4	0,6400
1	2	4	5	69,5	5	0,7478
1	2	4	6	59,8	6	0,8556
1	2	5	5	59,3	5	0,7297
1	2	5	6	59,5	6	0,8551
1	2	5	7	59,5	7	0,9801
1	2	6	6	59,8	6	0,8556
1	2	6	7	59,3	7	0,9797
1	2	6	8	59,3	8	1,1047
1	2	7	7	59,5	7	0,9801
1	2	7	8	59,5	8	1,1051
1	2	8	8	59,5	8	1,1051
1	3	1	2	219,5	2	0,6378
1	3	1	3	204,0	3	0,7354
1	3	1	4	189,8	4	0,8352
1	3	1	5	149,5	5	0,8891
1	3	1	6	160,0	6	1,0327
1	3	1	7	159,5	7	1,1568
1	3	1	8	169,5	8	1,2995
1	3	2	2	149,3	2	0,5137
1	3	2	3	124,0	3	0,5941
1	3	2	4	119,5	4	0,7111
1	3	2	5	109,8	5	0,8189
1	3	2	6	89,8	6	0,9086
1	3	2	7	89,8	7	1,0336
1	3	2	8	79,8	8	1,1409
1	3	3	3	104,0	3	0,5587
1	3	3	4	99,8	4	0,6762
1	3	3	5	89,5	5	0,7831
1	3	3	6	69,8	6	0,8732
1	3	3	7	59,5	7	0,9801
1	3	3	8	59,8	8	1,1056
1	3	4	4	79,8	4	0,6409
1	3	4	5	79,8	5	0,7659
1	3	4	6	69,8	6	0,8732
1	3	4	7	59,8	7	0,9806
1	3	4	8	60,0	8	1,1060
1	3	5	5	69,5	5	0,7478
1	3	5	6	59,5	6	0,8551
1	3	5	7	59,5	7	0,9801
1	3	5	8	49,5	8	1,0875

ctrl strat.	login				prod. time	costs	fitness
1	1	3	6	6	59,8	6	0,8556
1	1	3	6	7	59,8	7	0,9806
1	1	3	6	8	50,0	8	1,0883
1	1	3	7	7	49,8	7	0,9629
1	1	3	7	8	49,8	8	1,0879
1	1	3	8	8	49,8	8	1,0879
2	1	1	1	1	404,0	1	0,8388
2	1	1	2	2	265,0	2	0,7182
2	1	1	3	3	309,0	3	0,9209
2	1	1	4	4	329,0	4	1,0813
2	1	1	5	5	330,0	5	1,2080
2	1	1	6	6	329,0	6	1,3313
2	1	1	7	7	328,0	7	1,4545
2	1	1	8	8	308,0	8	1,5442
2	1	2	2	2	213,0	2	0,6263
2	1	2	3	3	214,0	3	0,7531
2	1	2	4	4	219,0	4	0,8869
2	1	2	5	5	227,0	5	1,0261
2	1	2	6	6	228,0	6	1,1528
2	1	2	7	7	237,0	7	1,2937
2	1	2	8	8	237,0	8	1,4187
2	1	3	3	3	191,0	3	0,7125
2	1	3	4	4	208,0	4	0,8675
2	1	3	5	5	209,0	5	0,9943
2	1	3	6	6	205,0	6	1,1122
2	1	3	7	7	218,0	7	1,2602
2	1	3	8	8	208,0	8	1,3675
2	1	4	4	4	183,0	4	0,8233
2	1	4	5	5	197,0	5	0,9731
2	1	4	6	6	192,0	6	1,0892
2	1	4	7	7	210,0	7	1,2460
2	1	4	8	8	190,0	8	1,3357
2	1	5	5	5	187,0	5	0,9554
2	1	5	6	6	196,0	6	1,0963
2	1	5	7	7	190,0	7	1,2107
2	1	5	8	8	200,0	8	1,3534
2	1	6	6	6	191,0	6	1,0875
2	1	6	7	7	189,0	7	1,2089
2	1	6	8	8	202,0	8	1,3569
2	1	7	7	7	187,0	7	1,2054
2	1	7	8	8	183,0	8	1,3233
2	1	8	8	8	192,0	8	1,3392
2	2	1	1	1	271,0	1	0,6038
2	2	2	2	2	253,8	2	0,6983

Note: the following three tables reproduce the three column-blocks of the rotated data table on this page. Each block has 44 rows and the same seven columns (listed here in the order they appear top-to-bottom in the image).

Block 1

col1	col2	col3	col4	col5	col6	col7
1,3675	8	208,0	8	8	1	3
0,6188	1	279,5	1	1	2	3
0,5839	2	189,0	2	2	2	3
0,6926	3	179,8	3	1	2	3
0,5141	2	149,5	2	2	2	3
0,5861	3	119,5	3	2	2	3
0,6758	4	99,5	4	2	2	3
0,5512	3	99,8	3	3	2	3
0,6581	4	89,5	4	3	2	3
0,7827	5	89,3	5	3	2	3
0,6400	4	79,3	4	4	2	3
0,7655	5	79,5	5	4	2	3
0,8732	6	69,8	6	4	2	3
0,7474	5	69,3	5	5	2	3
0,8728	6	69,5	6	5	2	3
0,9978	7	69,5	7	5	2	3
0,8556	6	59,8	6	6	2	3
0,9797	7	59,3	7	6	2	3
1,1047	8	59,3	8	6	2	3
0,9801	7	59,5	7	7	2	3
1,1051	8	59,5	8	7	2	3
1,1051	8	59,5	8	8	2	3
0,5663	2	179,0	2	1	3	3
0,7098	3	189,5	3	1	3	3
0,8348	4	189,5	4	1	3	3
0,9148	5	164,0	5	1	3	3
1,0225	6	154,3	6	1	3	3
1,1568	7	159,5	7	1	3	3
1,2995	8	169,5	8	1	3	3
0,5124	2	148,5	2	2	3	3
0,5861	3	119,5	3	2	3	3
0,6754	4	99,3	4	2	3	3
0,7734	5	84,0	5	2	3	3
0,8984	6	84,0	6	2	3	3
1,0336	7	89,8	7	2	3	3
1,1409	8	79,8	8	2	3	3
0,5508	3	99,5	3	3	3	3
0,6405	4	79,5	4	3	3	3
0,7734	5	84,0	5	3	3	3
0,8807	6	74,0	6	3	3	3
0,9801	7	59,5	7	3	3	3
1,1056	8	59,8	8	3	3	3
0,6405	4	79,5	4	4	3	3
0,7557	5	74,0	5	4	3	3

Block 2

col1	col2	col3	col4	col5	col6	col7
0,7359	5	62,8	5	5	3	2
0,9156	6	93,8	6	5	3	2
1,0128	7	78,0	7	5	3	2
1,1153	8	65,3	8	5	3	2
0,8609	6	62,8	6	6	3	2
1,0172	7	80,5	7	6	3	2
1,1197	8	67,8	8	6	3	2
0,9700	7	53,8	7	7	3	2
1,1396	8	79,0	8	7	3	2
1,1250	1	566,0	1	1	1	3
0,9461	2	394,0	2	1	1	3
0,7443	3	209,0	3	1	1	3
0,8675	4	208,0	4	1	1	3
0,9943	5	209,0	5	1	1	3
1,1175	6	208,0	6	1	1	3
1,2425	7	209,0	7	1	1	3
1,3657	8	207,0	8	1	1	3
0,9673	2	406,0	2	2	1	3
0,7425	3	208,0	3	2	1	3
0,8675	4	208,0	4	2	1	3
0,9907	5	207,0	5	2	1	3
1,1175	6	208,0	6	2	1	3
1,2443	7	209,0	7	2	1	3
1,3657	8	207,0	8	2	1	3
0,7443	3	209,0	3	3	1	3
0,8516	4	199,0	4	3	1	3
0,9925	5	208,0	5	3	1	3
1,1175	6	208,0	6	3	1	3
1,2425	7	209,0	7	3	1	3
1,3693	8	208,0	8	3	1	3
0,8675	4	208,0	4	4	1	3
0,9907	5	207,0	5	4	1	3
1,1175	6	208,0	6	4	1	3
1,2425	7	208,0	7	4	1	3
1,3675	8	208,0	8	4	1	3
0,9925	5	208,0	5	5	1	3
1,0981	6	197,0	6	5	1	3
1,2425	7	208,0	7	5	1	3
1,3675	8	208,0	8	5	1	3
1,1193	6	209,0	6	6	1	3
1,2425	7	208,0	7	6	1	3
1,3675	8	208,0	8	6	1	3
1,2248	7	198,0	7	7	1	3
1,3534	8	200,0	8	7	1	3

Block 3

col1	col2	col3	col4	col5	col6	col7
0,7562	3	215,8	3	1	1	2
0,4859	2	133,5	2	2	2	2
0,6595	3	161,0	3	2	2	2
0,7394	4	135,5	4	2	2	2
0,5592	3	104,3	3	3	2	2
0,7478	4	140,3	4	3	2	2
0,8105	5	105,0	5	4	2	2
0,6661	4	94,0	4	4	2	2
0,8101	5	104,8	5	4	2	2
0,9192	6	95,8	6	5	2	2
0,7730	5	83,8	5	5	2	2
0,9377	6	106,3	6	6	2	2
1,0256	7	85,3	7	7	2	2
0,8807	6	74,0	6	6	2	2
1,0353	7	90,8	7	7	2	2
1,1330	8	75,3	8	8	2	2
1,0062	7	74,3	7	7	2	2
1,1564	8	88,5	8	8	2	2
1,1321	8	74,8	8	8	2	2
0,6815	2	244,3	2	1	3	2
0,7323	3	202,3	3	1	3	2
0,8304	4	187,0	4	1	3	2
0,9505	5	184,3	5	1	3	2
1,1104	6	204,0	6	1	3	2
1,2191	7	194,8	7	1	3	2
1,3428	8	194,0	8	1	3	2
0,4863	2	133,8	2	2	3	2
0,6312	3	145,0	3	2	3	2
0,7116	4	119,8	4	2	3	2
0,8295	5	115,8	5	2	3	2
0,9532	6	115,0	6	2	3	2
1,0782	7	115,0	7	2	3	2
1,2027	8	114,8	8	2	3	2
0,5292	3	87,3	3	3	3	2
0,7005	4	113,5	4	3	3	2
0,8004	5	99,3	5	3	3	2
0,9196	6	96,0	6	3	3	2
1,0411	7	94,0	7	3	3	2
1,1497	8	84,8	8	3	3	2
0,6281	4	72,5	4	4	3	2
0,8149	5	107,5	5	4	3	2
0,8887	6	78,5	6	4	3	2
1,0102	7	76,5	7	4	3	2
1,1325	8	75,0	8	4	3	2

The fitness values of all 275 model variants are shown graphically in figure 6.9.
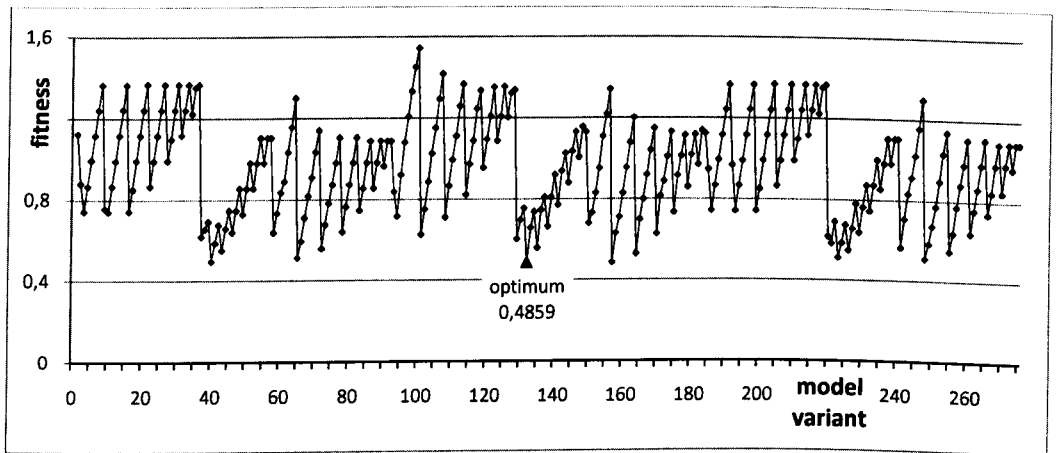
Figure 6.9 Fitness values of all variants with the optimum at X_{132}

The limits of the objective function parameters i.e. model generation and simulation results and objective function results are shown in table 6.5. The solution X_{132} has the minimal fitness value 0.4859 i.e. this solution is the global optimum. Figure 6.6 shows the PES and figure 6.7 the model structure of this variant.

	min	max
production time	49,5	566
costs	1	8
fitness	0.4859	1,5442

Table 6.5 Limits of fitness function parameters and results

Beside the global minimum several local minima exist with a very close fitness value, as can be seen in figure 6.9. Table 6.6 lists the global optimum (green line) and all near optimal solutions with a maximal variation of 3% of the maximal fitness value of 2. The solutions 2, 4 and 7 are identical to solutions 1, 3 and 6 due to the preferred assignment of the two available operators to the automatic login i.e. the manual login is not used in variants 2, 4, 7. The solutions 1, 3 and 6 differ in the control strategy whereas the most flexible control strategy 2 delivers the optimal result. Solutions 3 and 5 are based on different system configurations. With the used same weighting of production time and costs the solution 3 is the optimal solution, with a higher weighting of production time solution 5 would be a better variant.

no.	ctrl strat.	login typ	# of splicers	# of ops	prod. time	costs		fittness
1	1	2	2	2	139,8	2		0,4969
2	1	3	2	2	149,3	2		0,5137
3	2	2	2	2	133,5	2		0,4859
4	2	3	2	2	133,8	2		0,4863
5	2	3	3	3	87,3	3		0,5291
6	3	2	2	2	149,5	2		0,5141
7	3	3	2	2	148,5	2		0,5124

Table 6.6 Optimal and near optimal solutions

With other relevance factors r_1 and r_2 the optimal system configuration is different. E.g. without the consideration of costs two global optima with a production time of 49.5 exist (X_{86} and X_{267}). These solutions produce the specified number of orders in the shortest time.

In each of the two GA optimisation experiments the optimisation was repeated 50 times to estimate average values because of the stochastic nature of GA. Each optimisation experiment uses one stop criterion as described in section 6.4.2.

The results with average number of investigated individuals, optimum and near optima found are shown in table 6.7. The results show that the number of investigated individuals (194 and 102) is significant less than the number of all variants (275). The probability to find the optimal or near optimal solution is high (68% and 50%) but the finding is not guaranteed. Both, the number of investigated individuals and the finding probability depend highly on chosen GA parameters as can be seen when comparing the results of optimisation experiment 1 and 2 in table 6.7.

	Stop criterion 1 (uses weighted average change over 20 generation)	Stop criterion 2 (uses weighted average change over 5 generation)
Average number of investigated individuals to find an optimum	194	102
Optimum X_{132}	47%	21%
Near optimal results with max 3.2% error	21%	29%

Table 6.7 Results of 50 optimisation experiments

An example of the development of individual fitness values, best and average generation fitness during a single optimisation experiment is shown in figure 6.10. The diagram shows the fast convergence of the average fitness of the generations. After two generations each generation contains the optimal solution once in minimum and after the 7th generation the fitness value does not change anymore.
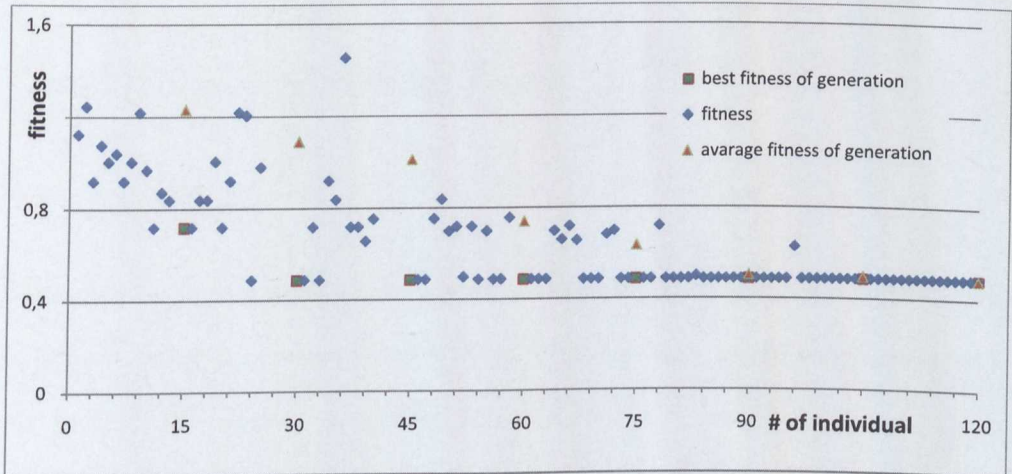
Figure 6.10 Individual fitness, best and average fitness of generations of one GA run

The results show that the optimisation approach developed in this research delivers an optimal solution with a high probability and with significantly less simulation runs in comparison to a complete simulation study of all model variants. Consequently the new approach of a simulation based parameter and structure optimisation is validated with a first real industrial example. There is a potential to increase the probability and/or decrease the number of simulation runs to estimate the optimal solution through adaptations of the GA parameters or with the use of other search methods.

For a potential application of the introduced approach it is necessary to extend the model to a complete Photofinishing Laboratory. Although the model of the case study is relative small the computing time of an optimisation experiment is on average between some 10 minutes and a few hours. However, the case study is carried out with a prototypical implementation of the simulation method and ideal parallelisation possibilities of GAs are not used. Hence, it can be assumed that there is a huge potential of runtime optimisation.

The introduced case study stands for many flexible production systems. It can be assumed that the developed framework can be applied to other, comparable systems with the ability of modular, hierarchical modelling.

Chapter 7

Conclusions and further Work

7.1 Conclusions

Simulation in a manufacturing context focuses on modelling the behaviour and the structure of manufacturing organisations, processes and systems. Many manufacturing systems have the potential to be optimised and to exploit this potential simulation based optimisation techniques are an important step forward. The overall goal of applying of these techniques is the identification of improved user selected system parameters. This research deals with a fundamental optimisation problem in discrete event simulation. Optimisation is well established but restricted to the optimisation of system parameters. Model structure is considered to be fixed, defined during model development. In simulation based optimisation using automated model parameter changes and manual model structure adaptations the global optimal system configuration cannot be guaranteed. With the growing use of flexible manufacturing systems and the increasing demand for product customisation the number of manufacturing system variants increases consequently the demand for structure optimisation is becoming increasingly more important.

This research has developed a simulation based optimisation method to solve the limitations of the established techniques. A crucial difference to established simulation based parameter optimisation is the application of a method based on meta-modelling to manage a set of models. The new optimisation method can simultaneously control both model

parameter changes and model structure selection. The result of a successful optimisation experiment using this approach is a parameter and structure optimised model. The key research aim to develop an approach to replace conventional manual structural changes i.e. to develop a combined, simulation based parameter and structure optimisation has been achieved.

An essential prerequisite of the new approach is a modular, hierarchical modelling and simulation method with a strict separation of model and simulator. This research determined the DEVS formalism as a suitable method. DEVS as a two-part definition consisting of a formal model specification and a simulation algorithm to model execution was introduced in the 70s and since then has been continuously developed. Many DEVS extensions have one joint attribute: they are based on the original DEVS formalism and have not taken advantage of the potential in combining extensions. For this reason the research has been followed the idea of a merging formalism to combine the advantages of different approaches. The new EDSDEVS formalism developed from this research is a fusion of Classic DEVS with selected extensions. It is an as generic as possible, powerful modelling and simulation formalism based on DEVS. A second key research aim to develop a modelling and simulation method based on DEVS and DEVS extensions to create a merging formalism has been achieved.

A further prerequisite for simulation based optimisation is an appropriate model management method. This research determined the SES/MB approach as a suitable method. Originally the SES/MB framework was developed to assist an analyst during a manual model variant selection. Changes to the SES/MB approach and algorithms to embed it into the simulation based optimisation have been developed within the research.

The final prerequisite is a suitable search method to find the optimal model configuration in the general multidimensional search space. Many search algorithms exist. One category widely used in both research and commercial applications are genetic and

evolutionary algorithms. For a practical investigation of the fundamental simulation based parameter and structure optimisation framework a commercial GA is used.

Validation of the work has been achieved using an industrial problem where the ability to control manufacturing system structure is an important optimisation factor. The photo-processing industry relies on management of the process flow to achieve profitability and this application demonstrates both how the new framework functions and the validity of the GA used in a real world situation. In two optimisation experiments it has been shown that the results are significantly dependent on the GA parameters. However in both experiments the probability to find an optimal or near-optimal model configuration is equal to or greater than 50%. An increased probability of an optimal solution is preferable however this will be the subject of further work.

The framework is implemented as MATLAB toolboxes and uses a commercial GA toolbox respectively. In the prototypical implementation of the framework and the validation of the work it has been shown that the use of MATLAB has both advantages and disadvantages. It is a powerful and productive environment to solve scientific and engineering problems and to implement prototypical applications. A disadvantage is the interpretative operation method. Particularly in simulation based optimisation where numerous, time consuming simulation runs lead to long execution times. However, there are parallel computing MATLAB toolboxes which support several aspects of parallelisation. The algorithmic summary shown using a GA is a promising approach to improve execution time by parallelisation.

During the research project the important steps have been published in a peer-reviewed journal, at international conferences and as a book chapter. Appendix C presents the publications.

7.2   Suggestions for further work

This research has established an approach to simulation based parameter and structure optimisation. Whilst this thesis presents the ideas, principles and a first example, it also opens up several future research directions. Future research directions can be divided into two areas (i) investigations of simulation based optimisation framework (ii) EDSDEVS formalism.

i.   The introduced approach defines the model structure variants at the meta-model level as a static structure. Otherwise it uses a dynamic structure modelling and simulation method to execute the selected model configuration. The dynamic changes of the model structure during the simulation time are not considered in this approach i.e. the optimisation regards only the initial model structure as a static structure. It seems feasible to add dynamic structure changes during the model lifetime as an additional criterion to the optimisation. An example is the length of stay of a sub model. This approach considers the initial existence of the sub model but its lifetime may play an important role in the search for an optimal model configuration.

With the SES XML definition a platform and implementation independent meta-modelling definition already exists. The manual modelling based on direct writing a XML file is not straightforward. General XML editors can assist the modelling but cannot replace a dedicated SES XML editor. A graphical SES/MB modelling application is a reasonable extension.

As already shown in section 6.4.3 the optimisation results and the number of optimisation cycles depends on the GA parameters. There is much literature about GA methods and parameterisation. The experience gained in this research has shown that further investigations in this direction are necessary. Hence, the optimisation of GA parameters is a further research topic.

There are also other promising search methods. Another nature analogue method is the Particle Swarm Optimisation (PSO) approach based on swarm intelligence of social groups. This group of algorithms is relative new, introduced around 10 years ago. The number of publications and applications is growing fast. The literature review has shown evidence that this algorithm group can solve problems like the simulation based optimisation as well as GAs.

ii. The new EDSDEVS formalism developed from this research is a fusion of Classic DEVS with several extensions. This part of the research is a step to a generic modelling and simulation formalism based on DEVS. Further extensions are desirable and essential e.g. extensions for parallel computing and graphical modelling. There are also approaches for hybrid DEVS extensions i.e. the support of continuous state changes. These are proposals for further research. The last proposal, the hybrid DEVS, is already a current research project topic of the Research Group CEA.

The importance and topicality of the idea behind this research can be seen in two brand new research proposals, the first currently in preparation and the second announced at 30.03.2009:

A research proposal at the Deutsche Forschungsgemeinschaft (DFG German Research Foundation) for further developments of the simulation based parameter and structure optimisation approach and its application to the optimisation of energy efficiency of process chains and manufacturing structures is currently in preparation. The optimisation of energy efficiency of process chains i.e. among other things the structure optimisation of process chains is a planned priority programme of DFG.

In a call for proposal of the Federal Ministry of Education and Research of Germany a sponsorship is announced with the topic 'safeguarding competitiveness by versatile manufacturing systems'. One matter of the proposed research is covered by the optimisation technique introduced in this thesis.

Appendix A.   References

[1]   Amnn W. (1994) *Eine Simulationsumgebung für Planung und Betrieb von Produktionssystemen.* Springer.

[2]   April J., Marco Better M., Glover F., Kelly J., Laguna M. (2006) *Enhancing Business Process Management with Simulaiton Optimization.* Proceedings of the 2006 Winter Simulation Conference.

[3]   April J., Kelly J., Glover F., Laguna M. (2003) *Practical Introduction to Simulation Optimization.* Proceedings of the 2003 Winter Simulation Conference.

[4]   April J., Glover F., Kelly J., and Laguna M. (2001) *Simulation/Optimization using "Real-World" Applications* Proceedings of the 2001 Winter Simulation Conference, pages 134-138.

[5]   Askin R.G., Standridge C.R. (1993) *Modeling an Analysis of Manufacturing Systems.* John Wiley & Sons.

[6]   Azadivar F. (1999) *Simulation Optimization Methodologies.* Proceedings of the 1999 Winter Simulation Conference, pages 93-100.

[7]   Banks J., Carson II J.S., Nelson B.L., Nicol D.M. (2003) *Discrete-Event System Simulation.* Prentice Hall.

[8]   Barnett M. (2003) *Modeling & Simulation in Business Process Management.* BP Trends Newsletter, White Papers & Technical Briefs, 1-10. http://www.bptrends.com [accessed November 20, 2008].

[9]   Barros F.J. (1996) *Modeling and Simulation of Dynamic Structure Discrete Event Systems: A General Systems Theory Approach.* PhD thesis. University of Coimbra.

[10]   Barros F.J. (1996) *The dynamic structure discrete event system specification formalism.* Transactions of The Society for Modeling and Simulation International, Mar 1996, vol. 13.

[11]   Breitenecker F. (1992) *Models, methods and experiments – A new structure for simulation systems.* Mathematics and Computer in Simulation 34, 1-30, Amsterdam: North Holland.

[12]   Chi S.D. (1997) *Model-based Reasoning Methodology Using the Symbolic DEVS Simulation.* Transaction of SCS 14(3) p.141-152.

[13]   Chow A.C., Zeigler B.P. (1994) *Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism.* Proceedings of the 1994 Winter Simulation Conference.

[14]   FU M.C., Glover F.W. (2005) *Simulation Optimization: A Review, New Developments, and Applications.* Proceedings of the 2005 Winter Simulation Conference.

[15]   Fu M. C., Andradóttir S., Carson J. S., Glover F., Harrell C. R., Yu-Chi Ho, Kelly J. P., Robinson S. M. (2000) *Integrating Optimization and Simulation: Research and Practise.* Proceedings of the 2000 Winter Simulation Conference.

[16]   Hagendorf O., Pawletta Th. (2008) *An Approach for Simulation Based Structure Optimisation of Discrete Event Systems.* Proceedings of the 2008 Spring Simulation Conference.

[17]   Hagendorf O., Pawletta T., Pawletta S., Colquhoun G. (2006) *An approach for modelling and simulation of variable structure manufacturing systems.* ICMR 2006 Liverpool/UK.

[18]   Hagendorf O., Colquhoun G., Pawletta T., Pawletta S. (2005) *A DEVS - Approach to ARGESIM Comparison C16 'Restaurant Business Dynamics' using MatlabDEVS.* Simulation News Europe, no.44/45, (December).

References

[19]   Heilala J., Montonen J., Salmela A., Järvenpää P. (2007) *Modeling and Simulation for Customer Driven Manufacturing System Design and Operations Planning.* Proceedings of the 2007 Winter Simulation Conference.

[20]   Holland J.H. (1975) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* The University of Michigan Press.

[21]   Ireson W.G. (1963) *Factory Planning and Plant Layout.* Prentice-Hall Englewood Cliffs, NJ.

[22]   Kalasky D.R. (1996) *Manufacturing Systems: Modeling and Simulation in Systems Modeling and Comupter Simulation.* Ed. Kheir N.A. Marcel Dekker, Inc.

[23]   Kremp M., Pawletta T., Colquhoun G. (2004) *Optimisation of manufacturing control strategies using online simulation.* In Proceedings of the 5th EUROSIM Congress on Modeling and Simulation, eds. G.Attiya & Y.Hamam (6 pages). Paris. SCS-European Publishing House, Ghent (Belgium).

[24]   Kremp M., Pawletta T., Pawletta S., Colquhoun G. (2004) *Simulation based control of a flexible manufacturing system, published in German.* Proc. of 11th symposium of maritime electrical engineering, electronics und information technology. (workshop: control and feedback control systems, pp. 15-18). University of Rostock.

[25]   Law A.M., Kelton W.D. (2000) *Simulation Modeling and Analysis.* McGraw-Hill 2000 3rd Edition.

[26]   Law A. M., McComas M. G. (1999) Simulation of Manufacturing Systems. Proceedings of the 1999 Winter Simulation Conference.

[27]   Lucie-Smith E. (1983) *A History of Industrial Design.* Phiadon Press Limited Oxford.

[28]   Maletzki G., Pawletta T., Pawletta S., Dünow P., Lampe B. (2008) *Simulation Model based Rapid Prototyping of Complex Robot Control Applications.* (German) In: atp-Automatisierungstechnische Praxis, Oldenbourg Verlag, München, 50(2008)8, pages 54-60.

[29]   Milberg J. (1992) *Wettbewerbsfaktor Zeit in Produktionsunternehmen.* (German) In: Tagungsband Münchner Kolloquium 91, Springer, pages 13-31.

[30]   Mittal S. (2007) *DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures.* PhD Thesis, University of Arizona.

[31]   Mittal S. (2007) *W3C XML schema Finite Deterministic DEVS Models.* http://www.saurabh-mittal.com/fddevs/ [accessed November 21, 2008].

[32]   Olafsson S., Kim J. (2002) *Simulation Optimization.* Proceedings of the 2002 Winter Simulation Conference.

[33]   Ören T. I. (1989) *Simulation Model: Taxonomy.* in: Encyclopaedia of Systems and Control, (Ed.) Singh, M., Pergamon Press.

[34]   Pawletta T., Deatcu C., Pawletta S., Hagendorf O., Colquhoun G. (2006) *DEVS-Based Modeling and Simulation in Scientific and Technical Computing Environments.* Proceedings of the 2006 Spring Simulation Conference, Huntsville/Al USA.

[35]   Pawletta T., Pawletta S. (2004) *A DEVS-based simulation approach for structure variable hybrid systems using high accuracy integration methods.* Proceedings of CSM2004 - Conference on Conceptual Modeling and Simulation, Part of the Mediterranean Modelling Multiconference (I3M), Genova, Italy, October 28-31 2004.

[36]   Pawletta T., Lampe B., Pawletta S., Drewelow, W. (2002) *A DEVS-Based Approach for Modeling and Simulation of Hybrid Variable Structure Systems.* Modeling, Anlysis, and Design of Hybrid Systems. Engel S., Frehse G., Schnieder E. (Ed.), Lecture Notes in Control and Information Sciences 279, Springer, pages 107-129.

[37]   Pawletta T., Pawletta S., Drewelow W. (1998) *Integration of discrete event simulation methods in interactive scientific and technical computing environments.*

In R. Zobel, editor, Proceedings of the 12th European Simulation Multiconference, pages 251-255. SCS European Publishing House, 1998. Manchester, June, 16-19.

[38] Pawletta T., Lampe B.P., Pawletta S., Drewelow W. (1996) *Dynamic structure simulation based on discrete events.* In ASIM-Mitteilungen Nr. 53, pages 7-11, 9. Workshop Simulation and AI, Ulm, Germany, Februar 1996.

[39] Pawletta T., Lampe B.P., Pawletta S., Drewelow W. (1996) *Modeling and Simulation of Variable Structure Systems.* Proc. of the 3rd International Symposium on Methods and Models in Automation and Robotics - MMAR'96 (IEEE), Miedzyzdroje, Poland, Ed.: Banka, S.; Domek, S. and Emirsajilow, Z.; 1996, Vol. 3, pages 1219-1223.

[40] Pawletta T., Lampe B.P., Pawletta S., Drewelow W. (1996) *A new Approach for Simulation of Variable Structure Systems.* Proceedings of the 41th Conference KoREMA (IEEE), Ed.: Vukic, Z.; Opatia, Croatia, September 1996 September 1996, Vol. 4, pages 83-87.

[41] Pawletta T., Lampe B.P., Pawletta S., Drewelow W. (1996) *An object oriented Framework for modeling and simulation of variable structure systems.* Proceedings of the SCS Summer Computer Simulation Conf., Portland, Oregon, July 1996, pages 8-13.

[42] Pawletta T., Pawletta S. (1995) *Design of a Simulator for Structure Variable Systems.* Proceedings of the 5th International IMACS-Symposium on System Analysis and Simulation, Berlin, SAMS 1995 Vol.18-19, Ed. Sydow, A., Gordon & Breach, 1995, pages 471-474.

[43] Pawletta T., Pawletta S. (1995) *Object-Oriented Simulation of Continuous Systems with Discrete Changes in Structure.* Proceedings of the 9th European Simulation Multiconference, Prag, Ed.: Snorek, M.; Sujansky, A. and Verbraeck, A., SCS International, 1995, pages 627-630.

[44] Pawletta T., Pawletta S. (1995) *Simulation of modular hierarchical systems with discrete structure changes.* Proceedings of the 40th Anniversary Conference KoREMA (IEEE), Zagreb, Ed. Vukic, Z.; April 1995, Vol. 1, pages 356 - 359.

[45] Pawletta T., Pawletta S., Dimitrov E. (1994) *Modelling and Simulation of Structure Variable Systems.* (German) Advances in Simulation (Fortschritte in der Simulationstechnik), Ed.: Kampe, G. and Zeitz, M., Vieweg Verlag, Braunschweig, 1994, pages 59-64.

[46] Pawletta T. (1992) *Comparison 2 - Modelling of a flexible manufacturing system, System EXTEND.* Simulation News Europe, (1992)6, pages 32-33.

[47] Pierreval H., Caux C., Paris J. L. , Viguier F. (2003) *Evolutionary approaches to the design and organization of manufacturing systems.* Computers and Industrial Engineering Volume 44, Issue 3 (March 2003).

[48] Praehofer, H. (1992) *CAST Methods in Modelling.* Pichler, F., Schwärtzel, H. Springer Pub.

[49] Ray J. P., Tomas S. C. (1998) *Simulation optimisation using a genetic algorithm.* Simulation Practice and Theory 6 (1998), pages 601–611.

[50] Rechenberg I. (1972) *Evolutionsstrategie.* (German) Friedrich Frommann Verlag.

[51] Rohrer M.W. (1998) *Simulation of Manufacturing and Material Handling Systems.* In: Handbook of Simulation ed. Banks J. John Wiley & Sons, Inc.

[52] Rozenblit J.W., Zeigler B.P. (1985) *Concepts for Knowledg--Based System Design Environments.* Proceedings of the 1985 Winter Simulation Conference.

[53] Sarjoughian H., Huang D. (2005) *A multi-formalism modeling composition framework: Agent and discrete-event models.* Paper presented at the 9th IEEE International Symposium on Distributed Simulation and Real Time Applications, Montreal, Quebec, Canada.

[54] Schönberg E., Heinzmann F., Feddersen S. (1994) *Genetic Algorithms and Evolutionary Strategies.* (German) Addison-Wesley.

[55]   Swisher J.R. (2003) *Discrete-Event Simulation Optimization using Ranking, Selection, and Multiple Comparison Procedures: A Survey.* ACM Transaction 04.2003.

[56]   Swisher, J.R. Hyden, P.D. (2000) *A Survey of Simulation Optimization Techniques and Procedures.* Proceedings of the 2000 Winter Simulation Conference.

[57]   The ACM Digital Library (2009) http://portal.acm.org.

[58]   The Mathworks™ (2008) *MATLAB™.* http://www.mathworks.com/products/matlab/.

[59]   The Mathworks™ (2008) *Genetic Algorithm and Direct Search Toolbox™.* http://www.mathworks.com/products/gads/.

[60]   Uhrmacher A.M., Arnold R. (1994) *Distributing and maintaining knowledge: Agents in variable structure environment.* 5th Annual Conference on AI, Simulation and Planning of High Autonomy Systems, pages 178-194.

[61]   Unified Modeling Language http://www.uml.org/ (2009).

[62]   Wainer, G. A. (2005) *DEVS Tools.* DEVSStandardization Group, http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm.

[63]   Wainer G., Giambiasi N. (2001) *Application of the Cell-DEVS paradigm for cell spaces modelling and simulation.* SIMULATION Transactions of The Society for Modeling and Simulation International, Jan 2001; vol. 76.

[64]   World Wide Web Consortium http://www.w3c.org/XML/ (2009).

[65]   Zeigler B. P., Hammonds P. E. (2007) *Modeling And Simulation-Based Data Engeneering.* Elsevier Academic Press.

[66]   Zeigler B.P., Praehofer H., Kim T.G. (2000) *Theory of Modelling and Simulation.* 3rd edition, Academic Press.

[67]   Zeigler B.P. (1984) *Multifacetted Modelling and Discrete Event Simulation.* Academic Press.

[68]   Zeigler B.P. (1976) *Theory of Modelling and Simulation.* 1st edition, John Wiley & Sons.

[69]   Zhang G., Zeigler B.P. (1989) *The system Entity Structure: Knowledge Representation for Simulation Modeling and Design.* In: Artificial Intelligence, Simulation, and Modeling. Widman L.E., Loparo K.A., Nielsen N.R. (Ed.), John Wiley & Sons Inc, pages 47-73.

Appendix B.   Coding Examples

atomic_model

   variables:

      t_{last} *time of last event*

      s   *internal state*

   function init()

      // initialise state variable set S and t_{next} with the time of the first internal event

   end function

   function $\delta_{ext}(e, x)$

      $t = t_{last} + e$

      // do something with x.value

   end function

   function $\delta_{int}(t)$

      $S_K \rightarrow S_{K+1}$ // calculate next internal state S_{K+1} from current internal state S_K

   end function

   function t = ta()

      $t = \dots$ // calculate next internal state event

   end function

   function $y = \lambda()$

      y.value = . . . // set value of y-message

   end function

end atomic_model

   Listing B.1 Pseudo code skeleton of an atomic Classic DEVS model

coupled_model

    function M_{d*} = select(imminent)

        M_{d*} = . . . // choose one of the sub component from component list imminent

    end function

end coupled_model

    Listing B.2 Pseudo code skeleton of a coupled Classic DEVS model

variables:

>   *t*   simulation clock*

>   *t_{end} simulation end time*

when receive start-msg(t_{end})

>   *send i-msg() to sub-ordinate DEVS coordinator*

>   *$t := t_{next}$ of sub-ordinate coordinator*

>   *while $t < t_{end}$*

>>   *send *-msg(t) to sub-ordinate DEVS coordinator*

>>   *$t := t_{next}$ of sub-ordinate coordinator*

Listing B.3 Pseudo code of a Classic DEVS root coordinator

variables:

t_{last} *time of last event*

t_{next} *time of next internal state event*

am   *associated atomic model*

when receive i-msg()

     am.init()

     $t_{last} := 0$

     $t_{next} := am.ta()$

when receive *-msg(t) at time t

     if $t <> t_{next}$

         error: bad synchronisation

     $y := am.\lambda()$

     send y in y-message to parent coordinator

     $am.\delta_{int}(t)$

     $t_{last} := t$

     $t_{next} := t_{last} + am.ta()$

when receive x-msg(t, x) at time t with value x

     if not $(t_{last} \leq t \leq t_{next})$

         error: bad synchronisation

     $am.\delta_{ext}(t-t_{last}, x)$

     $t_{last} := t$

     $t_{next} := t_{last} + am.ta()$

Listing B.4 Pseudo code of a Classic DEVS simulator

variables:

t_{last} *time of last event*

t_{next} *time of next internal state event*

CM   *associated coupled model*

when receive i-msg( )

     foreach sub component $M_d \in CM.M$

         send i-msg( ) to M_d

     $t_{last} := 0$

     // determine time of next scheduled internal state event of all sub components

     $t_{next} := min(\{ M_d.t_{next} \mid M_d \in CM.M \})$

when receive *-msg(t) at time t

     if $t <> t_{next}$

         error: bad synchronisation

     // find all sub components with a true condition $t_{next}=t$

     imminent := $\{ M_d \mid M_d \in CM.M \wedge M_d. t_{next} = t \}$

     // call select function to determin one sub component to send the *-msg

     $M_{d*} := select(imminent)$

     send *-msg(t) to M_{d*}

     $t_{last} := t$

     // determine time of next scheduled internal state event of all sub components

     $t_{next} := min(\{ M_d.t_{next} \mid M_d \in CM.M \})$

when receive x-msg(t, x) at time t with value x

     if not ($t_{last} \leq t \leq t_{next}$)

         error: bad synchronisation

     // get all sub components M_{d*} with an appropriate EIC

     receivers := subcomponents $\{M_d \mid M_d \in CM.M\}$ with $\{coupling \mid coupling \in CM.EIC\}$

// forwards the x-msg to all appropriate sub components

foreach sub component M_{d*} in receivers

    send x-msg(t, x) to M_{d*}

t_{last} := t

// determine time of next scheduled internal state event of all sub components

t_{next} := min({$M_d.t_{next}$ | $M_d \in CM.M$})

when receive y-msg(t, y) at time t with value y

    // forwards y-msg to super-ordinate model if an appropriate EOC exists

    if exist coupling in CM.EOC

        send y-msg(t, y) to parent model

    // get all sub components M_{d*} with an appropriate IC

    receivers := subcomponents {M_d |$M_d \in CM.M$} with {coupling | coupling $\in$ CM.IC}

    // creates from y-msg and sends it as an x-msg to all appropriate sub components

    foreach sub component M_{d*} in receivers

        send x-msg(t, y$\rightarrow$x) to M_{d*}

Listing B.5 Pseudo code of a Classic DEVS coordinator

atomic_model

    variables:

        t_{last} *time of last event*

        s *internal state*

    function $\delta_{ext}(e, x)$

        $t = t_{last} + e$

        switch x.port

            case $inputport_0$

                // do something with x.value received at input port $inputport_0$

                . . .

            case $inputport_n$

                // do something with x.value received at input port $inputport_n$

                . . .

        end switch

    end function

    function $y = \lambda()$

        y.port = . . . // set output port of y-message

        y.value = . . . // set value of y-message

    end function

Listing B.6 Pseudo code skeleton of an atomic Classic DEVS with Ports model

when receive *-msg(t) at time t

    if t <> t_{next}

        error: bad synchronisation

    y := am.λ()

    *send value y.value in y-message to parent coordinator **at port y.port***

    am.$\delta_{int}(t)$

    t_{last} := t

    t_{next} := t_{last} + am.ta()

*when receive x-msg(t, x, **p**) at time t with value x **at port p***

    if not ($t_{last} \leq t \leq t_{next}$)

        error: bad synchronisation

    *am.δ_{ext}(t-t_{last}, x, **p**)*

    t_{last} := t

    t_{next} := t_{last} + am.ta()

Listing B.7 Pseudo code of a Classic DEVS with Ports simulator

*when receive x-msg(t, x, **p**) at time t with value x **at port p***

    if not ($t_{last} \leq t \leq t_{next}$)

        error: bad synchronisation

    // get all sub components M_{d*} with an appropriate EIC

    receivers := subcomponents {M_d | $M_d \in$ CM.M} with {coupling | coupling $\in$ CM.EIC}

    // forwards the x-msg to all appropriate sub components

    foreach sub component M_{d*} in receivers

        *send x-msg(t, x, $M_{d*}.p$) to M_{d*} **at port p***

    t_{last} := t

    // determine time of next scheduled internal state event of all sub components

    t_{next} := min({$M_d.t_{next}$ | $M_d \in$ CM.M})

*when receive y-msg(t, y, **p**) at time t with value y **at port p***

    // forwards y-msg to super-ordinate model if an appropriate EOC exists

    if exit coupling in CM.EOC

        // coupling is a structure with the elements {sub component, p_{source}, $p_{destination}$}

        foreach coupling in CM.EOC

            *send y-msg(t, y, **coupling.$p_{destination}$**) to parent model*

    // get all sub components M_{d*} with an appropriate IC

    receivers := subcomponents {M_d | $M_d \in$ CM.M} with {coupling | coupling $\in$ CM.IC}

    // creates x-msg from y-msg and sends it as an x-msg to all appropriate

    sub components

    foreach sub component M_{d*} in receivers

        foreach coupling in CM.IC with coupling between y.source and $M_{d*}.p$

            *send x-msg(t, y$\rightarrow$x, $M_{d*}.p$) to M_{d*} **at port p***

        Listing B.8 Pseudo code of a Classic DEVS with Ports coordinator

atomic_model

    variables:

      t_{last} *time of last event*

      s   *internal state*

    function init()

      // initialise state variable set S and t_{next} with the time of the first internal state

        event

    end function

    function $\delta_{con}(t, x\_bag)$

      // default implementation of a confluent function matches Classic DEVS

        functionality

      $\delta_{int}(t)$

      $\delta_{ext}(0, x\_bag)$

    end function

    function $\delta_{ext}(e, x\_bag)$

      $t = t_{last} + e$

      foreach x in x_bag

        // do something with x.value

    end function

    function $\delta_{int}(t)$

      $S_K \rightarrow S_{K+1}$ *// calculate next internal state S_{K+1} from current internal state S_K*

    end function

    function t = ta()

      $t = \ldots$ *// calculate next internal state event*

    end function

    function $y\_bag = \lambda()$

y.value = . . . // set value of y-message

y_bag += y

end function

end atomic_model

Listing B.9 Pseudo code skeleton of an atomic PDEVS model

when receive *-msg(t) at time t

    if t <> t_{next}

        error: bad synchronisation

    y_bag := am.$\lambda()$

    send y_bag in y-message to parent coordinator

*when receive x-msg(t, **x_bag**) at time t with **x_bag***

    if not ($t_{last} \leq t \leq t_{next}$)

        error: bad synchronisation

    if t=t_{next} and x_bag is not empty

        // concurrent external and internal event

        am.δ_{con}(t, x_bag)

    else if t=t_{next} and x_bag is empty

        // internal event

        am.δ_{int}(t)

  else

        // external event

        am.δ_{ext}(t-t_{last}, x_bag)

  end if

  t_{last} := t

  t_{next} := t_{last} + am.ta()

Listing B.10 Pseudo code of a PDEVS simulator

atomic_model

variables:

t_{last} *time of last event*

s   internal state

function init(t)

// *initialise state variable set S and t_{next} with the time of the first internal state event*

// *t=0 initialisation at simulation start*

// *t>0 initialisation after structure change*

end function

function $\delta_{con}(t, x\_bag)$

// *default implementation of a confluent function matches Classic DEVS functionality*

$\delta_{int}(t)$

$\delta_{ext}(0, x\_bag)$

end function

function $\delta_{ext}(e, x\_bag)$

$t = t_{last} + e$

foreach x in x_bag

// *do something with x.value received at x.port*

switch x.port

case $inputport_0$

// *do something with x.value received at input port $inputport_0$*

. . .

case $inputport_n$

// *do something with x.value received at input port $inputport_n$*

. . .

        end switch

    end function

    function $\delta_{int}(t)$

        $s_u \rightarrow s_{u+1}$ *// calculate next internal state s_{u+1} from current internal state s_u*

    end function

    function $t = ta()$

        $t = \ldots$ *// calculate next internal state event*

    end function

    function $y\_bag = \lambda()$

        y.value = . . . // set value of y-message

        y.port = . . . // set output port of y-message

        y_bag += y

    end function

end atomic_model

Listing B.11 Pseudo code skeleton of an atomic EDSDEVS model

coupled_model

    variables:

        t_{last} *time of last event*

        s   *internal state*

    function init(t)

        // initialise structure and state variable set S and t_{next} with the time of the first internal

        // state event

        // t=0 initialisation at simulation start

        // t>0 initialisation after structure change

    end function

    function $\delta_{con}(t, x\_bag)$

        // default implementation similar to an atomic model

        functionality

        $\delta_{int}(t)$

        $\delta_{x\&s}(0, x\_bag)$

    end function

    function $\delta_{x\&s}(e, x\_bag)$

        $t = t_{last} + e$

        foreach x in x_bag

            // do something with x.value received at x.port

            switch x.port

                case $inputport_0$

                    // do something with x.value received at input port $inputport_0$

                  . . .

                case $inputport_n$

```
                    // do something with x.value received at input port inputport_n

        . . .

        end switch

    end function

    function  δ_int(t)

        s_u → s_{u+1}  // calculate next internal state s_{u+1} from current internal state s_u

    end function

    function  t = ta()

        t = . . .      // calculate next internal state event

    end function

    function  y_bag = λ(t)

        y.value = . . . // set value of y-message

        y.port = . . . // set output port of y-message

        y_bag += y

    end function

end coupled_model
```

Listing B.12 Pseudo code skeleton of a coupled EDSDEVS model

variables:

  t_{last} *time of last event*

  t_{next} *time of next internal state event*

  am *associated atomic model*

when receive i-msg(t) at time t

// t=0 initialisation at simulation start

// t>0 initialisation after structure change

  am.init(t)

  $t_{last} := t$

  $t_{next} := am.ta()$

when receive *-msg(t) at time t

  if t <> t_{next}

    error: bad synchronisation

  y_bag := am.λ()

  send y_bag in a y-message to parent coordinator

when receive x-msg(t, x_bag) at time t with value x_bag containing x.value und x.port pairs

  if not ($t_{last} \leq t \leq t_{next}$)

    error: bad synchronisation

  if t=t_{next} and x_bag is not empty

    // concurrent external and internal event

    am.δ_{con}(t, x_bag)

  else if t=t_{next} and x_bag is empty

    // internal event

    am.δ_{int}(t)

  else

    // external event

$am.\delta_{ext}(\ t\text{-}t_{last},\ x\_bag)$

end if

$t_{last} := t$

$t_{next} := t_{last} + am.ta()$

Listing B.13 Pseudo code of an EDSDEVS simulator

Coding Examples

variables:

t_{last} *time of last event*

t_{next} *time of next internal state event of the coupled model or a sub component*

t_{next_c} *time of next internal state event of the coupled model*

CM *associated atomic model*

       // CM.s_t current, sequential structure state

IMM   *// imminent children*

mail   *// output mail bag*

// t=0 initialisation at simulation start

// t>0 initialisation after structure change

when receive i-msg(t)at time t

     CM.init(t)

     foreach sub component $M_d \in CM.s_t.M$

        send i-msg(t) to M_d

     $t_{last} := t$

     // determine time of next scheduled internal state event of coupled model

     $t_{next\_c} := CM.ta()$

     // determine time of next scheduled internal state event of coupled model and all

     // sub components

     $t_{next} := min(t_{next\_c}, \{ M_d.t_{next} \mid M_d \in CM.s_t.M \})$

when receive *-msg(t) at time t

     if t <> t_{next} & t<>t_{next_c}

        error: bad synchronisation

     // internal state transition event of the coupled model CM itself

     if t=t_{next_c}

        y_bag := CM.λ()

[150]

send bag of value/output port pairs in a y-message to parent coordinator

// internal state transition event of a sub component of CM

else if $t=t_{next}$

// find all sub components with a true condition $t_{next}=t$

$IMM := \{ M_d \mid M_d \in CM.s_t.M \wedge M_d. t_{next}= t \}$

foreach M_d in IMM

send *-msg(t) to M_d

when receive x-msg(t, x_bag) at time t with value x_bag containing pairs of x.value/x.port

if not $(t_{last} \leq t \leq t_{next\_c})$

error: bad synchronisation

if $t=t_{next\_c}$ and x_bag is not empty

$CM.\delta_{con}(t, x\_bag)$ // concurrent external and internal event

else if $t=t_{next\_c}$ and x_bag is empty

$CM.\delta_{int}(t)$ // internal event

else

$CM.\delta_{x\&s}(t-t_{last}, x\_bag)$ // external event

end if

// get all sub components M_{d*} with an appropriate EIC

receivers := subcomponents $\{M_d \mid M_d \in CM.s_t.M\}$ with $\{coupling \mid coupling \in$

$CM.s_t.EIC\}$

// forwards the x-msg to all appropriate sub components

foreach sub component M_{d*} in receivers

$CM.\delta_{x\&s}(t-t_{last}, x\_bag)$ // external event of sub component

send x-msg(t, x_bag, $M_{d*}.p$) to M_{d*} at port p

foreach sub component M_{d*} in IMM and not in receivers

send x-msg(t, NULL, NULL) to M_{d*} // send empty bag, input port is ignored

$t_{last} := t$

$t_{next\_c} := t_{last} + CM.ta()$

$t_{next} := min(t_{next\_c}, \{ M_d.t_{next} \mid M_d \in CM.s_t.M \})$

when receive y-msg(t, y_bag, d) at time t with y_bag with value/port pairs from d

    // collect all y-messages from all sub components

    if d is not the last not reporting d in IMM

        add (y_bag, d) to mail

        mark d in IMM as reporting

    // all sub components now handled their *-message

    else if d is the last not reporting d in IMM

        $CM.\delta_{x\&s}(t\text{-}t_{last}, mail)$

        // check external coupling to form sub-bag of parent output

        $y\_bag_{parent} = NULL$

        foreach d in mail where (y_bag and d) has an appropriate EIC

            add y_bag to y_bag_{parent}

        send y-msg(t, y_bag_{parent}, CM) to parent model

        // check IC to get children M_{d*} with an appropriate IC who receives a sub bag

        receivers := subcomponents $\{M_d \mid d$ in mail, $M_d \in CM.s_t.M\}$ with {coupling |

            coupling $\in CM.s_t.IC\}$

        foreach sub component M_{d*} in receivers

            creates sub bag x_bag from mail with elements where M_{d*} is receiver

            send x-msg(t, x_bag) to M_{d*}

            mark d in IMM as sending

        foreach sub component M_{d*} in IMM where M_{d*} is not sending

            send x-msg(t, NULL) to M_{d*}

    $t_{last} := t$

    $t_{next\_c} := t_{last} + CM.ta()$

$$t_{next} := min(\ t_{next\_c},\ \{\ M_d.t_{next}\ |\ M_d \in CM.s_t.M\ \}\ )$$

Listing B.14 Pseudo code of an EDSDEVS coordinator

Coding Examples

```xml
<?xml version="1.0" encoding="us-ascii"?>
<!--
    DTD for an SES.
-->

<!ELEMENT top (ses_mb)>

<!ELEMENT ses (modelbase | ses | properties)*>

<!ELEMENT modelbase ((mb_composite | mb_atomic | mb_aspect |
mb_specialization | mb_specializationentity | mb_multiAspect)+)>

<!ELEMENT ses (composite)>

<!ELEMENT properties ((modelcouplings | var | varNumberOfComponent |
constraint)+)>

<!ELEMENT modelcouplings ((eic | eoc | ic)+)>
<!ATTLIST modelcouplings
        esname CDATA #REQUIRED>

<!ELEMENT mb_composite EMPTY>
<!ATTLIST mb_composite
        esname CDATA #REQUIRED>

<!ELEMENT composite ((aspect | specialization | multiAspect)*)>
<!ATTLIST composite
        esname CDATA #REQUIRED>

<!ELEMENT mb_atomic ((inports | outports)*)>
<!ATTLIST mb_atomic
        esname CDATA #REQUIRED
        classname CDATA #REQUIRED
        modelname CDATA #REQUIRED>

<!ELEMENT atomic EMPTY>
<!ATTLIST atomic
        esname CDATA #REQUIRED>

<!ELEMENT mb_aspect ((inports | outports)*)>
<!ATTLIST mb_aspect
        esname CDATA #REQUIRED
        classname CDATA #REQUIRED
        modelname CDATA #REQUIRED>

<!ELEMENT aspect ((entity | specialization | multiAspect |
atomic)*)>
<!ATTLIST aspect
        esname CDATA #REQUIRED>

<!ELEMENT mb_specialization ((inports | outports)*)>
<!ATTLIST mb_specialization
        esname CDATA #REQUIRED>

<!ELEMENT mb_specializationentity EMPTY>
<!ATTLIST mb_specializationentity
        esname CDATA #REQUIRED
```

```
        classname CDATA #REQUIRED
        modelname CDATA #REQUIRED>


<!ELEMENT specialization (specializationentity+)>
<!ATTLIST specialization
        esname CDATA #REQUIRED>


<!ELEMENT specializationentity EMPTY>
<!ATTLIST specializationentity
        esname CDATA #REQUIRED>


<!ELEMENT mb_multiAspect EMPTY>
<!ATTLIST mb_multiAspect
        esname CDATA #REQUIRED>


<!ELEMENT multiAspect (atomic)>
<!ATTLIST multiAspect
        esname CDATA #REQUIRED>



<!--
internal var will be set internally in the ses
external var references an external variable
-->
<!ELEMENT var EMPTY>
<!ATTLIST var
        name CDATA #REQUIRED
        esname CDATA #REQUIRED
        typ (internal|external) "internal"
        external_name CDATA #IMPLIED
        value CDATA #IMPLIED>


<!ELEMENT varNumberOfComponent EMPTY>
<!ATTLIST varNumberOfComponent
        esname CDATA #REQUIRED
        min CDATA #REQUIRED
        max CDATA #REQUIRED>


<!ELEMENT inports (inport+)>

<!ELEMENT outports (outport+)>

<!ELEMENT inport EMPTY>
<!ATTLIST inport
        name CDATA #REQUIRED>


<!ELEMENT outport EMPTY>
<!ATTLIST outport
        name CDATA #REQUIRED>


<!ELEMENT eic EMPTY>
<!ATTLIST eic
        inport CDATA #REQUIRED
        component CDATA #REQUIRED
        component_inport CDATA #REQUIRED>


<!ELEMENT eoc EMPTY>
<!ATTLIST eoc
        component CDATA #REQUIRED
```

```
      component_outport CDATA #REQUIRED
      outport CDATA #REQUIRED>

<!ELEMENT ic EMPTY>
<!ATTLIST ic
      component1 CDATA #REQUIRED
      component1_outport CDATA #REQUIRED
      component2 CDATA #REQUIRED
      component2_inport CDATA #REQUIRED>

<!ELEMENT constraint EMPTY>
<!ATTLIST constraint
      name CDATA #REQUIRED
      typ (entity|parameter) #REQUIRED
      action (enable|valid) #IMPLIED
      condition (gt|lt|eq|gteq|lteq|neq) #IMPLIED
      var_name1 CDATA #IMPLIED
      var_name2 CDATA #IMPLIED
      destination CDATA #IMPLIED>
```

Listing B.15 DTD describing the structure of SES/MB XML

Coding Examples

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ses SYSTEM "ses.dtd" []>
<ses_mb>
  <ses>
    <composite esname="ROOT">
      <aspect esname="ROOTdec">
        <composite esname="A">
          <specialization esname="Aspec">
                <specializationentity esname="A1"/>
                <specializationentity esname="A2"/>
          </specialization>
        </composite >
        <composite esname="B">
          <aspect esname="Bdec">
                <atomic esname="D"/>
                <atomic esname="E"/>
          </aspect>
        </composite >
      </aspect>
    </composite >
  </ses>

  <modelbase>
    <mb_aspect esname="ROOTdec" classname="ROOT" modelname="root"/>
    <mb_specialization esname="Aspec">
      <outports>
        <outport name="Aout1"/>
        <outport name="Aout2"/>
      </outports>
    </mb_specialization>
    <mb_aspect esname="Bdec" classname="B" modelname="b">
      <inports>
        <inport name="Bin1"/>
        <inport name="Bin2"/>
      </inports>
      <outports><outport name="Bout"/></outports>
    </mb_aspect>
    <mb_atomic esname="D" classname="D" modelname="d">
      <inports><inport name="Din"/></inports>
      <outports><outport name="Dout"/></outports>
    </mb_atomic>
    <mb_atomic esname="E" classname="E" modelname="e">
      <inports>
        <inport name="Ein1"/>
        <inport name="Ein2"/>
      </inports>
      <outports><outport name="Eout"/></outports>
    </mb_atomic>
  </modelbase>

  <properties>
    <modelcouplings esname="ROOTdec">
      <ic component1="A" component1_outport="Aout1"
          component2="B" component2_inport="Bin1"/>
      <ic component1="A" component1_outport="Aout2"
          component2="B" component2_inport="Bin2"/>
    </modelcouplings>
    <modelcouplings esname="Bdec">
      <eic inport="Bin1" component="D" component_inport="Din"/>
      <eic inport="Bin2" component="E" component_inport="Ein2"/>
```

Coding Examples

```
      <ic component1="D" component1_outport="Dout"
        component2="E" component2_inport="Ein1"/>
      <eoc component="E" component_outport="Eout" outport="Bout"/>
    </modelcouplings>
    <var esname="ROOT" name="pmax" typ="internal" value="6"/>
    <var esname="A1" name="p1" typ="internal" value="2"/>
    <var esname="A2" name="p1" typ="internal" value="3"/>
    <var esname="D" name="p2" typ="internal" value="3"/>
    <constraint name="sc1" condition="lt" var_name1="p1+p2"
              var_name2="pmax" action="valid" typ="parameter"/>
  </properties>
</ses_mb>
```

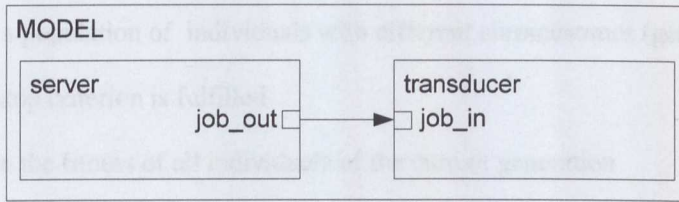Listing B.16 SES/MB XML example – XML file

[158]

Coding Examples

Figure B.1 A coupled model example

```xml
<?xml version="1.0" encoding="utf-8"?>
<atomic modelName="server" xmlns="AtomicDevs">
  <inports/>
  <outports>
    <outport>job_out</outport>
  </outports>
</atomic>


<?xml version="1.0" encoding="utf-8"?>
<atomic modelName="transducer" xmlns="AtomicDevs">
  <inports>
    <inport>job_in</inport>
  </inports>
  <outports/>
</atomic>
```

Listing B.17 Two atomic model XML files

```xml
<?xml version="1.0" encoding="utf-8"?>
<coupled modelName="MODEL" xmlns="CoupledDevs">
  <Models>
    <Model><devs>server</devs></Model>
    <Model><devs>transducer</devs></Model>
  </Models>
  <inports/>
  <outports/>
  <EIC/>
  <IC>
    <Coupling>
      <SrcModel>server</SrcModel><outport>job_out</outport>
      <DestModel>transducer</DestModel>
<inport>job_in</inport>
    </Coupling>
  </IC>
  </EOC>
</coupled>
```

Listing B.18 Coupled model XML file

0. Define the search space and chose an appropriate information encoding in chromosomes

1. Initialise a population of individuals with different chromosomes (generation 0)

Repeat until stop criterion is fulfilled

2. Estimate the fitness of all individuals of the current generation

3. Select pairs with *m* individuals and create descendants using crossover

4. Mutate the descendants

5. Exchange individuals of the current generation with descendants based on a substitution schema to create a new generation

<div align="center">Listing B.19 A general GA algorithm</div>

Appendix C.   Photofinishing Machines
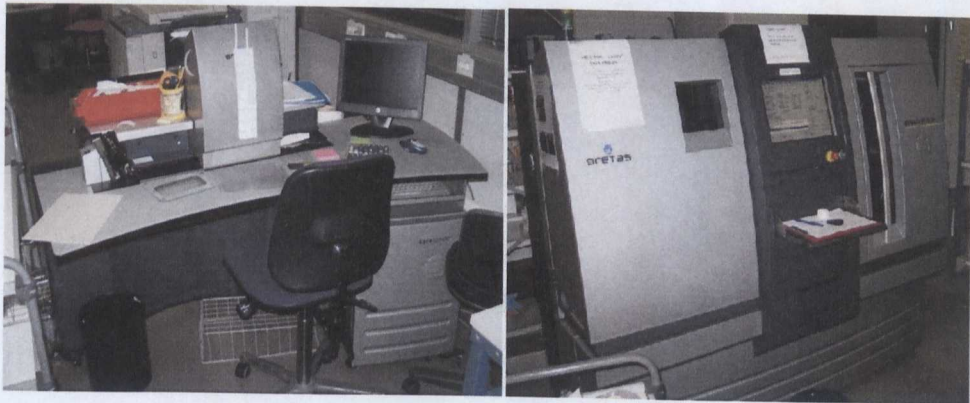

Figure C.1 Splicer (left) and URS

Figure C.2 DigiURS (left) and High-speed film scanner

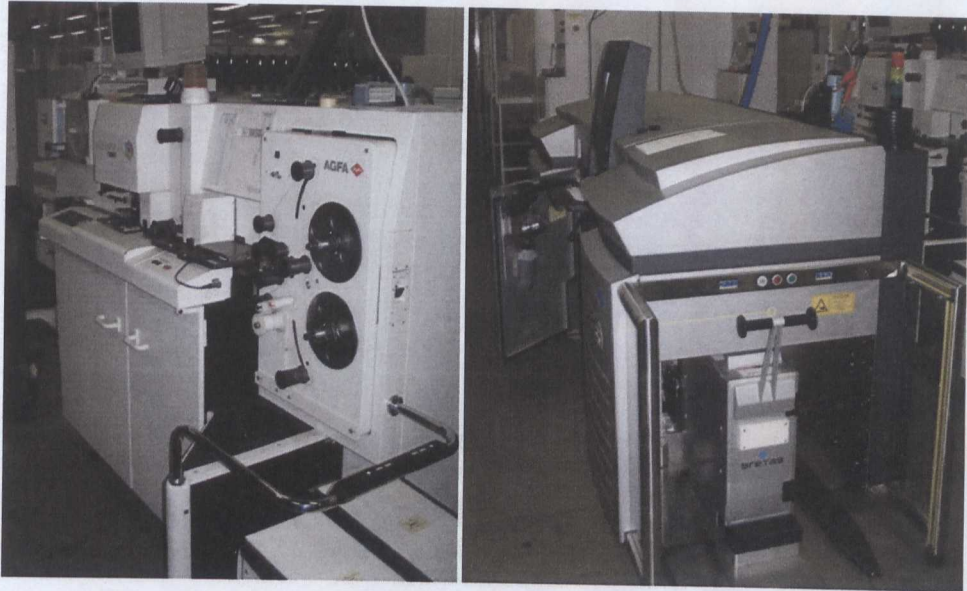Figure C.3 Analogue (left) and digital printer

Figure C.4 Manual (left) and automatic cutter